UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Diseño e implementación de un sistema de almacenamiento para entornos Fog

Máster Universitario en Ingeniería de Telecomunicación

Autor: LOZANO BAHÓN, Diego

Tutor: RAMOS DE SANTIAGO, Francisco Javier

Ponente: LÓPEZ DE VERGARA MÉNDEZ, Jorge E.

Dpto. Tecnología Electrónica y de las Comunicaciones

Diciembre, 2018

Diseño e implementación de un sistema de almacenamiento para entornos Fog

Autor: LOZANO BAHÓN, Diego
Tutor: RAMOS DE SANTIAGO, Francisco Javier
Ponente: LÓPEZ DE VERGARA MÉNDEZ, Jorge E.

Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Diciembre de 2018

RESUMEN

Actualmente, el progreso está marcado por el rápido desarrollo de Internet y la computación en la nube. Estas tecnologías han permitido la gestión de información ubicuamente, permitiendo a las personas la posibilidad de acceder a sus datos desde cualquier sitio y en cualquier momento. En la actualidad existen multitud de servicios gratuitos de almacenamiento en la nube, ofreciendo a los usuarios un *backup* de sus archivos, la posibilidad de compartirlos y gestionarlos ellos mismos. Además, estas empresas ofrecen la posibilidad de aumentar el espacio de almacenamiento con precios muy reducidos. Con motivo del desarrollo en los últimos años de la nube y del Internet de las cosas (*Internet of Things*), surge el concepto de *Fog Computing*, un modelo en el que los datos y el procesamiento se concentran en los dispositivos al borde de la red en lugar de *Cloud Computing*.

Debido a la evolución de *Internet of Things*, las aplicaciones demandan respuestas en tiempo real, lo que implica disminuir la latencia y un menor consumo de ancho de banda por la proximidad de los dispositivos. Es por ello por lo que surge la necesidad de desarrollar el concepto de *Fog Computing*

En este sentido, en este trabajo de fin de máster se propone el desarrollo e implementación de un sistema de almacenamiento *Fog* similar a los ya existentes en la nube. En este sistema, un usuario comparte un archivo distribuyéndolo entre otros dispositivos cercanos, respetando en todo momento la privacidad del usuario y de sus archivos. Además se ha implementado un mecanismo de *blockchain* para almacenar el historial y los cambios de los archivos así como la ubicación de los documentos compartidos. De esta manera, se conoce en todo momento la información de cada archivo y cada servidor de forma segura, ya que un bloque de la cadena de bloques no puede ser modificado una vez añadido.

Palabras clave: nube computacional, niebla computacional, Internet de las cosas, seguridad, encriptado, distribuido, cadena de bloques.

ABSTRACT

Progress is determined by the fast development of Internet and cloud computing. These technologies have allowed the management of information ubiquitously, allowing people to access data from anywhere and at any time. Currently there are many storage services in the cloud, offering users backup for their files and the ability to share and manage them by themselves. In addition, these companies offer the possibility of increasing storage at very low prices. Due to the recent development of the cloud and the Internet of Things, the concept of Fog Computing emerges, like a model in which data and processing are located in the edge devices instead of Cloud Computed

Due to the evolution of the Internet of Things, applications demand answers in real time, which means reducing latency and reducing bandwidth consumption due to the proximity of devices. For that reason emerges the need to develop the concept of Fog Computing.

That is why this project is proposing the development and implementation of a Fog system, in which the user shares a file distributing it among other close devices, respecting at all times the privacy of the users and their files. Also, a blockchain mecanism has been implemented to store location, changes and records of the files. In this way, information of each file and each server is known any time in a safe way, since a block from the blockchain cannot be modified once added.

Keywords: Cloud Computing, Fog Computing, Internet of Things, security, encrypt, distributed, blockchain.

Agradecimientos.

Me gustaría dar las gracias a todas aquellas personas que me han ayudado a llegar hasta el final de la carrera y a hacer este proyecto de fin de Master, pero me gustaría destacar a algunas:
Primero a mis padres y hermana que siempre me han apoyado en todo lo posible, especialmente en los momentos más difíciles.
A Javier Ramos, mi tutor de la universidad, por su tiempo y dedicación durante todos estos meses.
A todos mis profesores a lo largo del Master por su paciencia y dedicación.
Gracias a mis amigos, por aguantarme en el día a día y por intentar entenderme en cada momento.
A mis compañeros de clase por estos dos años estudiando juntos y por los viajes que hemos hecho.
A mis compañeros de trabajo, por permitirme estudiar a la vez que trabajaba e irme siempre antes de tiempo.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este primer capítulo se presenta el contenido de este proyecto y se indica la motivación para la realización de este Trabajo de Fin de Máster. Además, se exponen los objetivos del trabajo, el plan de trabajo y la organización de esta memoria.

1.1. MOTIVACIÓN

A día de hoy, existe un creciente uso de la nube, debido a la posibilidad de acceder a contenido compartido en cualquier momento y desde cualquier dispositivo de manera segura y protegiendo el contenido o la capacidad de compartir información. A pesar de ello, todavía hay algunos problemas sin resolver, como la falta de determinación en la latencia, la falta de soporte en la movilidad y la posibilidad de mover parte de los procesos al cliente o dispositivos intermedios [1]. En los últimos años ha surgido con fuerza la idea de *Fog Computing* que propone permitir directamente la comunicación entre los nodos más externos de la red, de manera que se puedan ofrecer nuevas aplicaciones y servicios para el futuro de Internet, en vez de trabajar con una estructura piramidal clásica. [2]

Se puede definir un entorno *Fog* como: "*un escenario en el que una gran cantidad de dispositivos heterogéneos, ubicuos y descentralizados se comunican y potencialmente cooperan entre ellos para realizar tareas de almacenamiento y procesamiento sin la intervención de terceros. Estas tareas pueden ser útiles para respaldar funciones básicas de red o nuevos servicios y aplicaciones que se ejecutan en un entorno aislado. La idea principal es que los usuarios puedan arrendar parte de sus dispositivos para alojar estos servicios a cambio de incentivos para hacerlo*." [2]

Un entorno *Fog* es una extensión de los servicios ofrecidos por la nube con el objetivo de disminuir la latencia con nodos distribuidos geográficamente de manera que permita la movilidad, la interacción en tiempo real y evitar la congestión en la red a la vez que aumenta la resiliencia. [3] En un entorno *Fog* o *Cloud* elásticas, los nodos hijos son dispositivos portátiles como por ejemplo *Smartphones*, sensores Wifi u otro tipo de dispositivos IoT. [3] [4]

A día de hoy, estos dispositivos están expuestos a posibles ataques maliciosos y fallos, lo cual provocaría una indisponibilidad de los servicios y recursos. Es por ello, fundamental que los contenidos estén replicados en varias localizaciones para que, en caso de ataque o fallo de uno de ellos, se puedan recuperar los datos originales. [3] [5]. Este hecho es particularmente importante en entornos *IoT* donde la información recolectada no puede perderse, como por ejemplo en el caso de los contadores inteligentes de electricidad. Por otro lado, al repartir la información entre múltiples dispositivos, los datos de los usuarios no se comparten en claro, sino que se deben encriptar a nivel de contenido y a nivel de transporte, de manera que el usuario que generó un contenido sea el único que pueda acceder a él. [6] El creciente uso de herramientas como Dropbox o Drive para compartir archivos, invita a utilizar nodos distribuidos en vez de centralizados, disminuyendo la latencia y aumentando el ancho de banda.

En esta línea, este trabajo presenta una implementación de un sistema de ficheros distribuido que proporciona seguridad y funcionalidad en entornos *Fog*.

Este TFM está relacionado con la asignatura de Tecnologías y servicios de internet, estudiada en el Máster de Ingeniería de Telecomunicaciones. En ella se estudió el concepto de *Cloud* privada y la herramienta de *Devstack.*

1.2. OBJETIVOS Y PLANTEAMIENTO

El objetivo de este Trabajo de Fin de Máster en Ingeniería de Telecomunicaciones es el diseño e implementación de un sistema de almacenamiento para entornos *Fog*. Para ello se plantean los siguientes objetivos:

- Desarrollar programas cliente y servidor en Java que intercambien archivos entre ellos.
- Proporcionar seguridad es un aspecto importante que debemos tener en cuenta. Se debe proporcionar un nivel de seguridad alto. Tanto las comunicaciones como los datos que se intercambien deben ir cifrados.

- Desarrollar un historial de los archivos compartidos mediante la tecnología *blockchain*, con información de los bloques y los servidores en los que se almacenan los datos, así como las modificaciones
- Emular escenarios realistas mediante el uso de la herramienta *Mininet*
- Comprobar mediante emulaciones las mejoras de latencia y seguridad de *Fog Computing* frente a *Cloud Computing*.

1.3. METODOLOGÍA Y PLAN DE TRABAJO

Como punto de partida se realizará un estudio previo del TFM, investigando y documentando sobre la situación de *Fog*. A continuación:

- Se ejecutarán tareas de investigación y documentación sobre el uso de sockets, *Transport Layer Security* (TLS), *blockchain* y distintos tipos de cifrado.
- Se desarrollará el código Java necesario para el envío y recepción de archivos.
- Una de las características más significativas de *Fog Computing* es la distribución de archivos en los distintos nodos, para ellos se implementará una funcionalidad de particionado de los datos entre los nodos disponibles.
- Implementación del programa con cifrado AES y TLS
- Para tener registrado el historial de cada uno de los nodos, se implementará un *blockchain*, en el cual se almacenará información de cada uno de los fragmentos en los que se ha dividido el archivo original y el nodo en el que se encuentra así como sus cambios.
- Implementación para el borrado de archivos, que elimina los archivos del nodo local y de los nodos en los que se ha distribuido.
- Se realizará una batería de pruebas con el fin de encontrar errores y subsanarlos. Para ello se probará tanto con operaciones que el programa no se espera, como con determinados tipos de archivos. Todos estos errores se corregirán y se controlarán.
- Por último, se analizará el tráfico con la herramienta *Wireshark*.

1.4. ORGANIZACIÓN DE LA MEMORIA

La organización de la memoria consta de los siguientes capítulos:

- Capítulo 1: Introducción. Este capítulo contiene la motivación, los objetivos de este proyecto y la organización de la memoria.

- Capítulo 2: Estado del arte. Esta sección analizaremos las tecnologías como el *Cloud Computing*, *Internet of Things* y *Fog Computing*, además se añade una sección con las aplicaciones prácticas más importantes. Se definen las herramientas utilizadas para la realización de este proyecto
- Capítulo 3: Diseño. Se definen cada una de las funciones básicas del programa y los pasos a seguir que debe hacer el usuario para cada una de las acciones.
- Capítulo 4: Desarrollo. Se explica paso a paso en que consiste cada una de las funciones del programa y la metodología que se ha llevado a cabo para el intercambio de archivos y los mecanismos de seguridad que se han utilizado para proteger la información.
- Capítulo 5: Integración, pruebas y resultados. Se explica la batería de pruebas que se ha llevado a cabo y el contexto de cada una de las pruebas comparando distintos parámetros con distintos tipos de archivos.
- Capítulo 6: Conclusiones y trabajo futuro. En este capítulo, después de analizar las pruebas, se comprueba si se han cumplido los objetivos marcados. Se definirán las ventajas e inconvenientes encontrado en este proyecto y las posibles mejoras que se pueden llevar a cabo en un futuro.

2. ESTADO DEL ARTE

En esta sección se analizarán los fundamentos básicos sobre los que se construye este trabajo, así como los trabajos previos similares. En concreto nos centraremos en tecnologías como el *Cloud Computing*, *Internet of Things*, *Fog Computing* y *blockchain*. Adicionalmente se describen las aplicaciones prácticas más importantes, así como las herramientas utilizadas para la realización de este proyecto.

2.1. CLOUD COMPUTING

A pesar de que el concepto de *Cloud Computing* parece bastante reciente, en la década de los 50 se empezaron a asentar las bases de esta tecnología. Por esa época, las grandes empresas tenían la necesidad de consultar gran cantidad de información desde distintos puntos, por lo que se empezó a estudiar la forma de integrar una CPU con múltiples usuarios. [7]

Se puede dar una primera definición de nube como un conjunto de servicios disponibles en la red que ofrece una infraestructura de computación de bajo coste, con una alta calidad de servicio (QoS), personalizada y por la cual se puede acceder de forma simple a los contenidos. [8]

En la década de los 90, la red de internet ya tenía un ancho de banda suficiente para soportar la nube, por lo que se empezó a desarrollar el proyecto. En 2002, ingenieros de Amazon desarrollaron un sistema de almacenamiento en la nube AWS (*Amazon Web Services*), que definieron como "*ofrece un conjunto completo de servicios de infraestructuras y aplicaciones que le permiten al usuario, ejecutar prácticamente todo en la nube, desde aplicaciones empresariales y proyectos de grandes datos hasta juegos sociales y aplicaciones móviles*". [7]

En 2006 surge el concepto de *Elastic Compute Cloud* (EC2) que ofrece un servicio de alquiler de servidores que permite funcionar sus propias aplicaciones a pequeñas y medianas empresas.

Debido al éxito que tuvo, empresas como Google o IBM empezaron a investigar sobre el *Cloud Computing*. [9]

En 2009 surge Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) como una infraestructura de *open source*, que permitía la creación de nubes privadas compatible con Amazon E2C. [9]

2.1.1. Aplicaciones de *Cloud computing*

Los servicios más populares de la nube son infraestructura como servicio, IaaS (*Infrastructure as a Service)*, plataforma como servicio, PaaS (*Platform as a Service*), software como servicio, SaaS (*Software as a Service*).

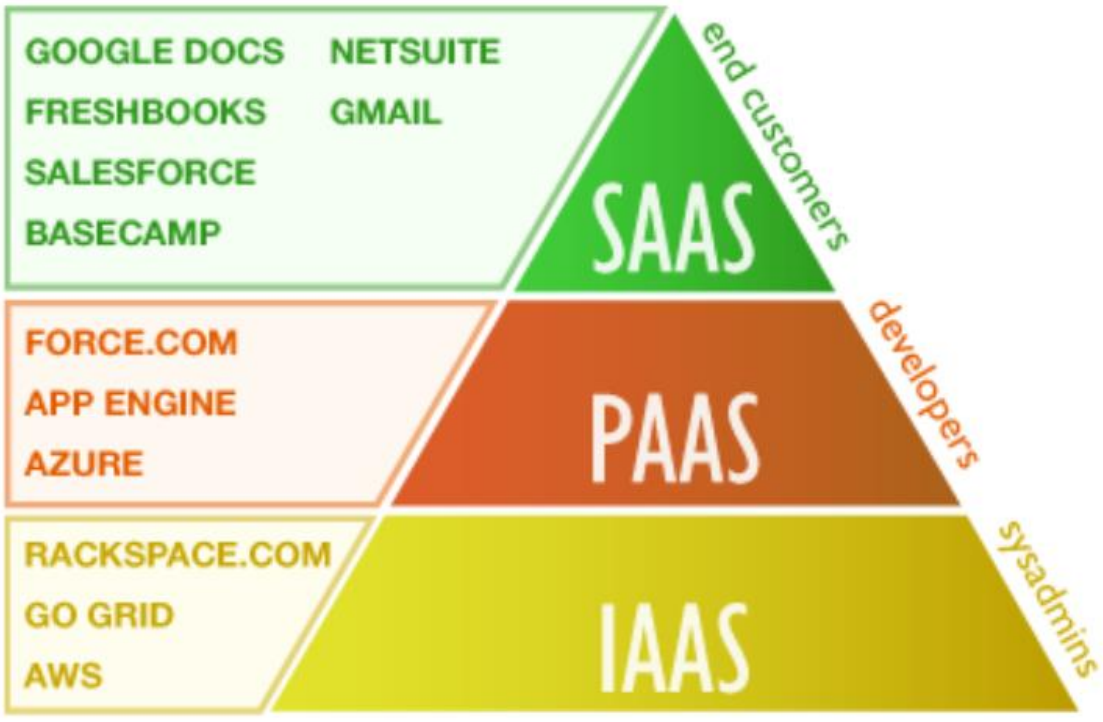

Fig. 1 Tipos de servicios en la nube y ejemplos [10].

- Infraestructura como servicio, IaaS (*Infrastructure as a Service*), proporciona los elementos básicos de la computación en la nube, como el procesamiento, almacenamiento y otras formas de recursos de hardware de forma virtual y bajo demanda a través de Internet. A diferencia de los servicios de alojamiento tradicionales como los servidores físicos los cuales, se alquilan mensualmente o anualmente, la infraestructura de la nube alquila máquinas virtuales en

función del uso y puede escalarse de manera dinámica, según las necesidades del cliente. Dicha escalabilidad y demanda se habilita por los recientes avances en la virtualización y la administración de la red. Los usuarios de IaaS no necesitan administrar o controlar la infraestructura de la nube subyacente, pero tienen control sobre los sistemas operativos, el almacenamiento, las aplicaciones implementadas y, en algunos casos, el control limitado de componentes de red seleccionados. [12] [11]

- Plataforma como servicio, PaaS (*Platform as a Service*), avanza un paso más que IaaS al proporcionar entornos de programación y ejecución al usuario. Un producto PaaS actúa como una plataforma integrada de diseño, desarrollo, prueba e implementación. El usuario de PaaS puede crear aplicaciones utilizando lenguajes de programación y API compatibles con el proveedor, y luego implementarlas directamente en la infraestructura de la nube del proveedor. El usuario de PaaS no administra ni controla la infraestructura de la nube subyacente, pero tiene control sobre las aplicaciones implementadas y la configuración del entorno de host de aplicaciones. Esto puede reducir la mayor parte de la carga de administración del sistema como la configuración y los cambios entre los distintos entornos de desarrollo, pruebas y productivo, que tradicionalmente llevan los desarrolladores, quiénes pueden concentrarse en problemas más productivos. Por lo general, PaaS proporciona un conjunto completo de herramientas de desarrollo, desde el diseño de la interfaz hasta la lógica del proceso y la integración. [12] [13]
- Software como servicio, SaaS (*Software as a Service*), proporciona a los usuarios aplicaciones completas a través de Internet, incluso sistemas complejos como los de CRM o ERP. El software o las aplicaciones se alojan como servicios en la nube y se entregan a través de navegadores una vez suscritos por el usuario. De esta forma se puede eliminar la necesidad de instalar, ejecutar y mantener la aplicación en las computadoras locales. SaaS es conocido por su arquitectura multiusuario en la que todos los usuarios comparten la misma base de código única mantenida por el proveedor. Las políticas de seguridad de autenticación y autorización se utilizan para garantizar la separación de los datos del usuario. Un mecanismo de intercambio de este tipo, permite que el costo y el precio de SaaS sigan siendo competitivos en comparación con el software tradicional. Se espera que SaaS alivie la carga del usuario del mantenimiento del software y reduzca el gasto de las compras de software a precios de demanda. [12] [14]

Fig. 2 Comparativa entre SaaS, PaaS y IaaS [11].

2.1.2. Tipos de Clouds

- Una nube privada está dentro del centro de datos empresarial de una organización. Las mayores ventajas que ofrece son las facilidades de administrar la seguridad, el mantenimiento, las actualizaciones y un mayor control sobre el desarrollo y su uso. Las nubes privadas pueden compararse con la intranet. A diferencia de las nubes públicas, donde todos los recursos y aplicaciones son gestionados por el proveedor del servicio, en las nubes privadas todos estos servicios se agrupan y se ponen a disposición de los usuarios a nivel organizativo. Los recursos y aplicaciones son gestionados por la propia organización. La seguridad se mejora, ya que sólo usuarios de las organizaciones tienen acceso a la nube privada. [15] [16]
- Nube de la comunidad: Cuando muchas organizaciones construyen y comparten una infraestructura en la nube, sus requisitos y políticas se denominan nube comunitaria. La infraestructura en la nube podría ser alojada por un proveedor externo o dentro de una de las organizaciones de la comunidad. [15] [16]
- La nube pública permite el acceso de los usuarios a la nube a través de interfaces mediante navegadores web. Los usuarios deben pagar sólo por el tiempo que usan el servicio, es decir, el pago por uso. Se paga en función del uso, esto ayuda a reducir los costos de operación en gastos de TI. Sin embargo, las nubes públicas, son menos seguras en comparación con otros modelos de nube, porque que todas las aplicaciones y datos en la nube pública son más propensos a los ataques maliciosos. La solución a esto puede ser que los controles de seguridad se implementen a través de la validación en ambos lados, tanto por el proveedor de la nube como por el cliente. [15] [16]
- Nube híbrida: Es una combinación de nube pública y nube privada. En este modelo, una nube privada está vinculada a uno o más servicios de nube externos. Es una forma más segura de

controlar los datos y las aplicaciones y le permite a la parte acceder a la información a través de Internet. Permite a la organización satisfacer sus necesidades en la nube privada y, si se produce alguna necesidad ocasional, solicita a la nube pública recursos informáticos intensivos. [15] [16]

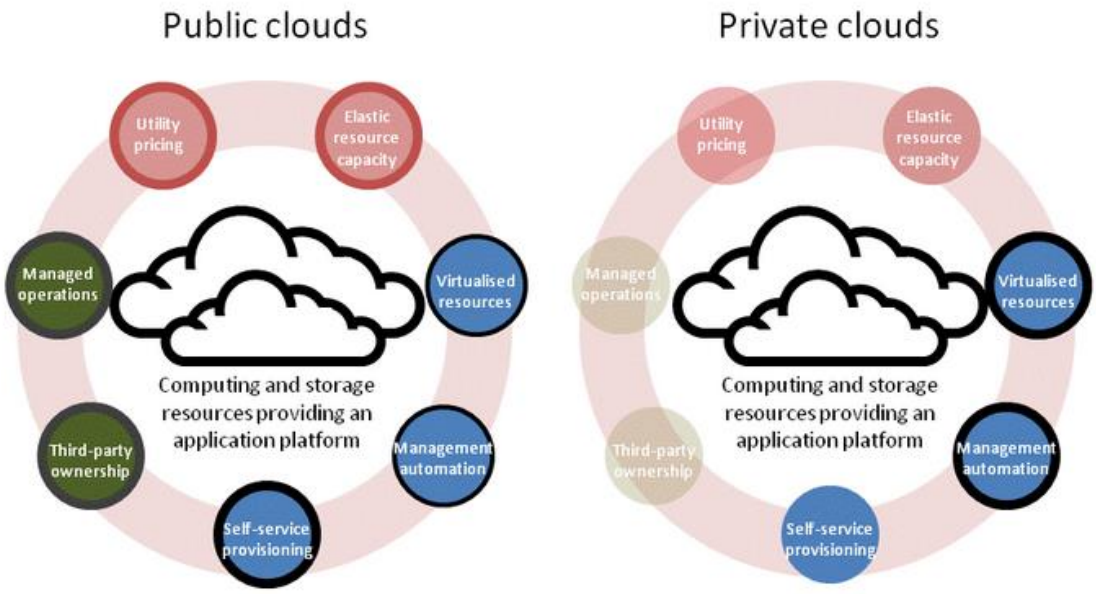

Fig. 3 Comparativa Cloud público y Cloud privado [10].

2.2. INTERNET OF THINGS

Internet of Things, IoT, se refiere a la interconexión en red de objetos cotidianos, que a menudo están conectados mediante redes inalámbricas y que pueden contener sensores. IoT aumentará la ubicuidad de Internet al integrar cada objeto para la interacción a través de sistemas integrados, lo que conduce a una red altamente distribuida de dispositivos que se comunican con seres humanos y otros dispositivos. Gracias a los rápidos avances en las tecnologías subyacentes, IoT está abriendo enormes oportunidades para un gran número de aplicaciones novedosas que prometen mejorar la calidad de nuestras vidas. En los últimos años, IoT ha ganado mucha atención de investigadores y profesionales de todo el mundo. [17] [18]

El concepto de Internet de las cosas fue propuesto por Kevin Ashton en el Auto-ID Center del MIT en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y tecnologías de sensores. [17]

La innovación de *Internet Of Things* se caracteriza por la combinación de componentes físicos y digitales para crear nuevos productos y habilitar nuevos modelos de negocio. Gracias a la eficiente

administración de energía, comunicación de ancho de banda, memoria y avance de las tecnologías de los microprocesadores, es posible digitalizar los productos. [19]

En consecuencia, se está desarrollando una gama de oportunidades para que las empresas generen un valor incremental en *Internet of Things*. La Figura 4 ilustra la lógica de tal creación de valor. Demuestra que las soluciones IoT suelen combinar cosas físicas con TI en forma de hardware y software. Como resultado, las funciones básicas de un objeto se pueden mejorar con servicios digitales adicionales basados en TI, a los que se puede acceder no sólo a nivel local sino a nivel global. Por ejemplo, la función primaria de una bombilla es proporcionar luz en una ubicación específica. Sin embargo, si la bombilla se mejora con la tecnología IoT, también puede detectar la presencia humana y servir como un sistema de seguridad de bajo coste que, en caso de intrusión, activa un modo de luz intermitente y envía una alerta al teléfono inteligente del propietario. Del mismo modo, la función principal de una papelera es proporcionar capacidad de almacenamiento. Pero cuando el contenedor se enriquece con la tecnología IoT, puede medir y monitorizar su propio peso, por lo tanto, detectar niveles bajos de stock y ofrecer un servicio de reabastecimiento automático. Y aunque la función primaria de un tractor puede ser remolcar otros equipos agrícolas, una conexión del tractor al IoT podría facilitar los servicios de optimización predictiva y mantenimiento basados en TI. [19]

THING	+	**IT** (Hardware / Software)	=	**Thing-based function** (Physical / Local)	+	**IT-based service** (Digital / Global)
	Bulb	IoT stack	→	Light		Security
	Bin	IoT stack	→	Storage capacity		Automatic replenishment
	Tractor	IoT stack	→	Towing vehicle		Predictive maintenance, optimization

Fig. 4 Ejemplos Internet of Things [19].

2.3. FOG COMPUTING

La primera vez que se utilizó el término de *Fog Computing* fue por Flavio Bonomi, D*istinguished System Engineer* (DSE) de Cisco. El nombre hace referencia al objetivo principal del *Fog* que es extender la funcionalidad del *Cloud* (almacenamiento, redes, procesamiento y recursos de

computación) al extremo de la red. Al igual que pasa con las nubes cuando bajan al suelo, convirtiéndose en niebla. [20]

Fog Computing se puede entender como un complemento al *Cloud Computing*, ya que sin él, *Fog* no tendría sentido, debido a la necesidad de un almacenamiento y procesado de información a más alto nivel. Es una plataforma virtualizada que extiende al *Cloud Computing* y los servicios al extremo de la red. Es semejante al *Cloud Computing* ya que ofrece almacenamiento de recursos de datos, redes, computación y aplicaciones a los usuarios finales, pero se diferencia en que los servicios y aplicaciones se encuentran en el extremo de la red, en los dispositivos de los usuarios, con el fin de minimizar su latencia y aumentar la calidad de servicio, algo muy importante en las aplicaciones industriales de *Internet of Everything* que requiere la información en tiempo real. [18]

Debido a la distribución geográfica de la información no centralizada, como en el *Cloud, Fog Computing* facilita el análisis de datos en tiempo real, *Big Data*, y añade un nuevo axioma de recopilación distribuida de datos. [21]

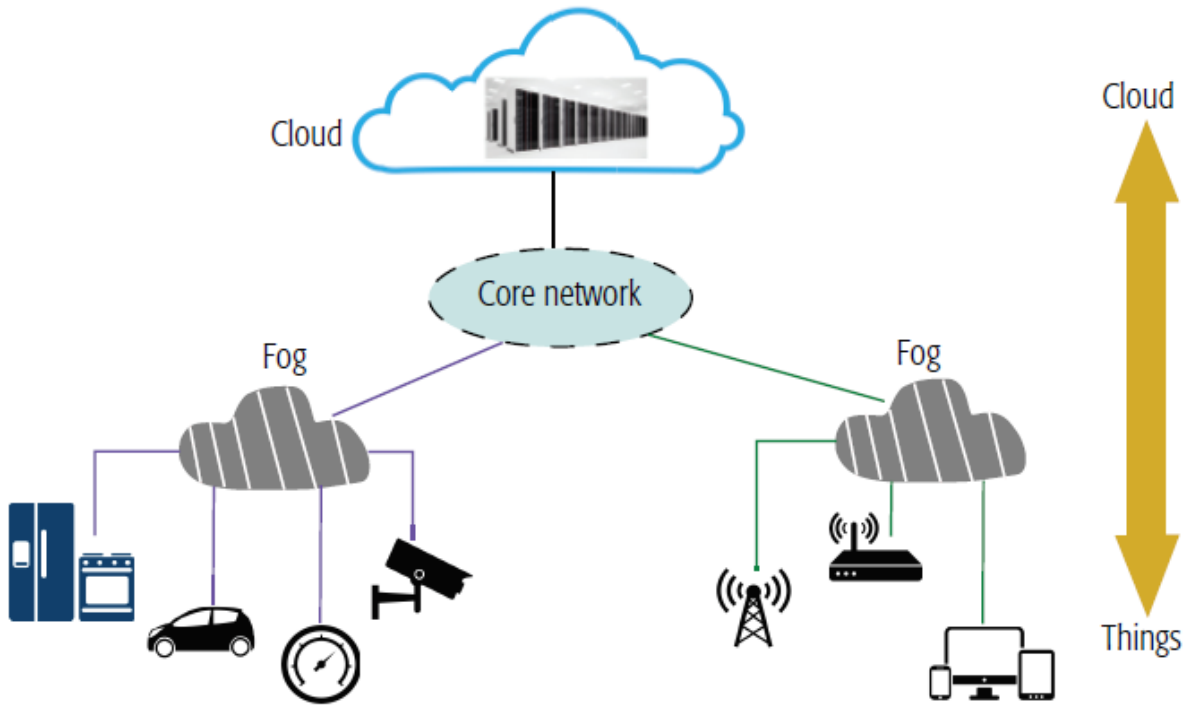

Fig. 5 Arquitectura de un sistema Fog [22].

El concepto de *Fog Computing* es un elemento imprescindible en el mundo de *Internet of Everything*, caracterizado por los millones de conexiones entre objetos, procesos, datos y personas. Este proceso generaría un enorme volumen de datos que deben ser procesados, analizados y convertidos en

información útil. La utilidad de esta información requiere que la información sea procesada en tiempo real, por lo que enviar esta información a la nube para ser procesada, supondría una elevada latencia. [20]

Cisco, uno de los principales participantes en el desarrollo de *Fog Computing*, cree que afectará a diversas aplicaciones de *Internet of Everything*, principalmente a *Smart Grids* (redes eléctricas inteligentes), vehículos conectados y *Smart Cities*, así como a redes de sensores y actuadores en general (WSAN, *Wireless Sensors and Actuators Networks*), y por tanto, puede aplicarse a los procesos de redes industriales capaces de monitorizar explotaciones petrolíferas y de gas, distribución, *retail* y trasporte de mercancías y viajeros. [19]

2.3.1. Características principales

- Ubicación en los extremos de la red, conocimiento de la ubicación y baja latencia. Los orígenes de *Fog Computing* se pueden rastrear hasta las primeras propuestas para respaldar los *endpoints* con servicios en el borde de la red, incluidas las aplicaciones con requisitos de baja latencia (por ejemplo, juegos, transmisión de video, realidad aumentada). [20]
- Distribución geográfica. A diferencia de la nube más centralizada, los servicios y aplicaciones a los que se dirige *Fog* exigen implementaciones ampliamente distribuidas. *Fog* tendrá un papel importante en la transmisión de información en los vehículos conectados a través de proxys o puntos de acceso localizados en las autopistas [20]
- Las redes de sensores a gran escala para monitorizar el medioambiente y la red inteligente son otros ejemplos de sistemas distribuidos que requieren recursos distribuidos de computación y almacenamiento. [20]
- Gran número de nodos, como consecuencia de la amplia geo-distribución, que encontramos en las *Smart Grid* con redes de sensores
- Soporte para movilidad: Para muchas aplicaciones de *Fog Computing*, es esencial comunicarse directamente con dispositivos móviles y, por lo tanto, utilizar protocolos como LISP, que desacoplan la identidad del host de la identidad de ubicación y requieren un sistema de directorio distribuido. [20]
- Interacciones en tiempo real: Las aplicaciones importantes de *Fog Computing* implican interacciones en tiempo real en lugar de procesamiento por paquetes.
- Predominio del acceso inalámbrico. [20]
- Heterogeneidad: en los nodos de una *Fog* pueden darse en distintas estructuras y se implementarán en una amplia variedad de entornos. [20]

- Interoperabilidad y federación: El soporte sin interrupciones de ciertos servicios requiere la cooperación de diferentes proveedores. Por lo tanto, los componentes de *Fog* deben ser interoperables, y los servicios deben estar federados en todos los dominios. [20]
- Soporte para el análisis en línea y la interacción con la nube: El *Fog* está posicionado para desempeñar un papel importante en la gestión y el procesamiento de los datos cerca de la fuente. [20]

2.3.2. Usuarios de Fog Computing

No es fácil determinar en esta etapa inicial cómo se alinearán los diferentes usuarios de *Fog Computing*. Sin embargo, según la naturaleza de los principales servicios y aplicaciones, anticipamos que: [20]

- Los modelos de suscriptores desempeñarán un papel importante en la niebla (*Infotainment* en vehículos conectados, *Smart Grid, Smart Cities, Health Care*, etc.)
- *Fog Computing* dará lugar a nuevas formas de competencia y cooperación entre los proveedores que buscan proporcionar servicios globales. Los nuevos titulares entrarán en la arena como usuarios y proveedores, incluidos los servicios públicos, fabricantes de automóviles, administraciones públicas y agencias de transporte.

2.4. APLICACIONES DE FOG COMPUTING E IOT

En esta sección vamos a explicar tres ejemplos de aplicaciones de *Fog Computing* e I*nternet of Things*:

2.4.1. Vehículos conectados

La implementación del vehículo conectado nos permite distintos tipos de interacciones: interacciones entre vehículos, interacciones entre vehículos y puntos de acceso (Wi-Fi, 3G, LTE, paneles informativos, semáforos) e interacciones entre distintos puntos de acceso. [20]

Con esta información, *Fog Computing* ofrece un servicio completo de seguridad, información y análisis del tráfico, así como su distribución en ciudades y carreteras o el conocimiento de movilidad, ubicación y soporte para interacciones en tiempo real. [20]

Los semáforos inteligentes, debido a los sensores, detectan peatones y ciclistas, midiendo la velocidad y las distancias a las que se aproximan. También permite la interacción con otros semáforos, permitiendo modificación en sus ciclos en función del tráfico o para evitar un accidente. [20]

La información obtenida por cada uno de los sensores es utilizada para realizar análisis en tiempo real y se enviar a la nube para su análisis a largo plazo para posibles mejoras. [20]

2.4.2. Smart Grid

Mientras que los extremos de *Fog Computing* proporcionan localización, lo que permite una baja latencia, la nube proporciona una centralización global. Muchas aplicaciones necesitan la localización del *Fog* y la globalización del *Cloud* especialmente para su análisis y *Big Data*. [20]

Los sensores que captan la información en los extremos de la red se relación con bucles de protección y requieren un procesamiento en tiempo real. [20]

En este primer nivel de *Fog*, se produce una interacción máquina a máquina (M2M), que recopila, procesa los datos y emite comandos de control a los actuadores. [20]

El segundo y tercer nivel tratan de visualización e informes, interacciones entre máquina y humano (HMI), así como de sistemas y procesos M2M. [20]

Las escalas de tiempo de estas interacciones, varían continuamente (análisis en tiempo real) e incluso, cada día (análisis transaccional). Como resultado, *Fog* debe soportar varios tipos de almacenamiento, desde el nivel más bajo hasta el nivel más alto. También observamos que, cuanto más alto es el nivel, más amplia es la cobertura geográfica y más larga es la escala de tiempo. La cobertura global, proporcionada por la nube, se utiliza como repositorio de datos, que tienen una permanencia de meses y años, y que son las bases para los negocios de inteligencia analítica. Este es el entorno de HMI típico de los informes y cuadros de mando de los indicadores de rendimiento clave de la pantalla. [20]

2.4.3. Wireless Sensors and Actuators Networks

Los WSNs (*Wireless Sensor Nodes*) fueron diseñados para operar con una potencia mínima para extender al máximo la vida útil de las baterías. La mayoría de los WSNs se caracterizan por reducido consumo de ancho de banda, energía y potencia de procesamiento. Los sensores ambientales, el procesamiento simple y el envío de datos al receptor estático son los principales objetivos de estas redes de sensores, que usan el estándar llamado TinyOS2. [20]

Los WSNs han demostrado ser útiles recopilando datos ambientales como: humedad, temperatura, cantidad de lluvia o intensidad lumínica. [20]

Las WSN con restricciones de energía avanzaron en varias direcciones: se propusieron múltiples sumideros, sumideros móviles, múltiples sumideros móviles y sensores móviles para cumplir con los requisitos de las nuevas aplicaciones. Sin embargo, son escasos en aplicaciones que van más allá de la detección y el seguimiento, pero requieren actuadores para ejercer acciones físicas (abrir, cerrar, mover, enfocar, orientar, incluso transportar y desplegar sensores). Los actuadores, que pueden controlar un sistema o el proceso de medición en sí, aportan nuevas dimensiones a las redes de sensores. [20]

El flujo de información no es unidireccional (desde los sensores hasta el receptor), sino bidireccional (desde los sensores hasta el receptor y desde el controlador al actuador). De esta manera, se convierte en un sistema de ciclo cerrado, en el cual, los problemas de estabilidad y comportamiento oscilatorio no pueden ser ignorados. La latencia y la inestabilidad se vuelven una preocupación dominante en los sistemas que requieren una respuesta rápida. [20]

Las características de *Fog* (conciencia de proximidad y ubicación, geo distribución, organización jerárquica) lo convierten en la plataforma adecuada para dar soporte a WSN y WSAN con restricciones de energía. [20]

2.5. COMPARACIÓN CLOUD COMPUTING, FOG COMPUTING E INTERNET OF THINGS

REQUISITO	CLOUD COMPUTING	FOG COMPUTING
LATENCIA	Alta	Baja
DEMORA EN LOS JITTER	Alta	Muy baja
UBICACIÓN DEL SERVICIO	Dentro de Internet	Al borde de las redes locales
DISTANCIA ENTRE CLIENTE-SERVIDOR	Múltiples saltos	Salto único
SEGURIDAD	Indefinida	Definida
ATAQUE EN DATOS ENRUTADOS	Alta probabilidad	Muy baja probabilidad
DEPENDENCIA DE LOCALIZACIÓN	No	Si
GEO DISTRIBUCIÓN	Centralizada	Distribuida
NUMERO DE SERVIDORES	Bajo	Muchos
SOPORTE A LA MOVILIDAD	Limitada	Soportada
INTERACCIÓN EN TIEMPO REAL	Soportada	Soportada
TIPO DE CONECTIVIDAD ÚLTIMA MILLA	Línea dedicada	Inalámbrica

Tabla I Comparativa Cloud Computing y Fog Computing [23].

REQUISITO	FOG COMPUTING	CLOUD COMPUTING	INTERNET OF THINGS
USO	Dispositivos móviles	Usuarios de internet	Dispositivos móviles y estaciones
NUMERO DE SERVIDORES	Alto	Bajo	Alto
ARQUITECTURA	Distribuida	Centralizada	Densa y distribuida
TIPO DE SERVICIO	Servicios de información localizados limitados a una ubicación de implementación específica	Información global recopilada en todo el mundo	Información específica de un dispositivo final
ENTORNO DE TRABAJO	Al aire libre o interior	Interior con espacios masivos y ventilación	Al aire libre e interior
CONOCIMIENTO DE LA UBICACIÓN	Si	No	Si
INTERACCIONES EN TIEMPO REAL	Soportado	Soportado	Soportado
MOVILIDAD	Soportada	Limitada	Soportada
BIG DATA Y DURACIÓN DE ALMACENAJE	Corta duración ya que se envía a Big Data	Meses y años	Transitorio, ya que es la fuente de Big Data
PROVEEDOR	Cisco iox	Amazon, Microsoft, IBM	ARM, Atmel, Bosh

Tabla II Comparativa Fog Computing, Cloud Computing y Internet of Things [3].

Características de las soluciones de niebla frente a la nube, resultados y comportamientos esperados:

CLOUD COMPUTING	FOG COMPUTING
Los datos y las aplicaciones se trabajan en la nube, para grandes volúmenes de información su procesamiento puede ser demorado.	*Fog* opera en el borde de la red de los dispositivos, en esta capa se consume menos tiempo de procesamiento.
El ancho de banda usado para el canal de la nube puede verse comprometido	Menos demanda de ancho de banda pues los datos son agregados en algún dispositivo intermedio antes de ser transportado a la nube, de ser preciso hacerlo.
Demoras en el tiempo de respuesta y problemas de escalabilidad debido a la localización de los servidores en el despliegue de la solución.	Por medio de servidores de borde determinados por la visibilidad de los usuarios en la red, se pueden fomentar menos demoras en los tiempos de respuesta finales y menores problemas de escalabilidad del despliegue

Tabla III Comparativa Cloud Computing y Fog Computing [23].

2.6. MININET

Se trata de un emulador de red que crea redes de *hosts* virtuales, *switches*, controladores y enlaces. Se utiliza la virtualización ligera para hacer que un solo sistema parezca una red completa. Es, hasta ahora, la plataforma de pruebas de red de código abierto que proporciona apoyo a la investigación de las SDN más conocida. [25]

En los *hosts* virtuales, se pueden ejecutar las aplicaciones de red basadas en Linux. En una red *OpenFlow* simulada con Mininet, una aplicación de controlador real *OpenFlow* se puede ejecutar en la misma máquina o en un equipo externo que simule los *host* virtuales. Esta herramienta utiliza el *kernel* de Linux y otros recursos para simulan los componentes de la SDN como los *host*, el controlador y los *switches OpenFlow*. [26]

Las topologías definidas son: *linear*, *single*, *tree* y personalizada. En todas ellas hay un controlador, varían en el número de *hosts*, *switches* y los enlaces entre estos:

1. *Single*: Se lanzará un *switch* conectado a N *hosts*

   ```
   sudo mn -- topo single, N
   ```

2. *Linear*: Se trata de una topología en la que cada *switch* se conecta con otro de forma lineal, y cada *switch* tiene un *host* conectado.

   ```
   sudo mn --topo linear,K,
   ```

3. *Tree*: Se creará una topología en árbol con profundidad N y anchura M.

   ```
   sudo mn --topo tree,depth=N,fanout=M
   ```

4. Personalizada: Para este tipo de topologías es necesario crear un archivo *Python* con su descripción. Las topologías de este tipo se quedarán guardadas con extensión .py.

Adicionalmente, Mininet permite la definición de topologías personalizadas mediante el uso de una API en *Python* que es de gran utilidad a la hora de automatizar pruebas y despliegues.

2.7. ENCRIPTACIÓN

Advanced Encryption Standard (AES) es uno de los algoritmos de cifrado más utilizados y seguros actualmente disponibles. Es de acceso público, y es usado por instituciones como la NSA. Su éxito empezó en 1997 cuando NIST (*National Institute of Standards and Technology*) buscaba una

alternativa de cifrado a DES. Se denomina el algoritmo "Rijndael", ya que fue desarrollado por Daemen y Rijmen, que destacaba por su flexibilidad, seguridad y rendimiento. [27].

AES es un cifrado simétrico que procesa datos en bloques de 128 bits. Admite tamaños de clave de 128, 192 y 256 bits y consta de 10, 12 o 14 rondas de iteración, respectivamente. Cada ronda mezcla los datos con una clave redonda, que se genera a partir de la clave de cifrado. El descifrado invierte las iteraciones que dan como resultado una ruta de datos parcialmente diferente. [28]

2.8. TLS

El protocolo SSL (*Secure Sockets Layer*) es un protocolo que está diseñado para permitir comunicaciones seguras entre aplicaciones. Para lograr esta seguridad, utiliza criptografía asimétrica y certificados X.509.

2.8.1. Funciones del protocolo SSL/TLS

- Seguridad criptográfica: TLS nos permite establecer una conexión segura entre dos entidades. Una conexión encriptada de TLS requiere que toda la información enviada entre un cliente y un servidor sea encriptada por el software de envío y descifrada por el software receptor, proporcionando así un alto grado de confidencialidad. La confidencialidad es importante para ambas partes en cualquier transacción privada. [29]
- Interoperabilidad: permite que dos entidades sean compatibles independientemente del ámbito de programación que se use. De esta manera, dos aplicaciones distintas pueden intercambiar información con éxito parámetros criptográficos sin tener conocimiento del código de la otra entidad. [29]
- Extensibilidad: TLS proporciona un marco estable, por lo que no es necesario crear nuevos protocolos o implementar librerías de seguridad. [29]
- Eficiencia: TLS incorpora un cache de sesiones con el fin de reducir en número de conexiones que se establecen. [29]

El protocolo de registro TLS se utiliza para la encapsulación de varios protocolos de nivel superior. Uno de estos protocolos encapsulados, el TLS *Handshake Protocol*, permite que el servidor y el cliente se autentiquen entre sí y negocien un algoritmo de cifrado y claves criptográficas antes de que el protocolo de la aplicación transmita o reciba su primer byte de datos. [29]

Cuando un cliente y un servidor TLS comienzan a comunicarse, acuerdan una versión de protocolo, seleccionan algoritmos criptográficos, se autentican de forma opcional y usan técnicas de cifrado de clave pública para generar secretos compartidos.

El Protocolo de TLS *Handshake* implica los siguientes pasos: [29]

- Intercambiar mensajes de saludo para acordar algoritmos, intercambiar valores aleatorios y verifique la reanudación de la sesión.
- Cambiar los parámetros criptográficos necesarios para permitir que el cliente y el servidor acuerden un secreto premaster.
- Certificados de intercambio de información criptográfica para permitir que el cliente y el servidor se autentiquen.
- Generar un secreto principal del secreto premaster e intercambiar valores aleatorios.
- Proporcionar parámetros de seguridad a la capa de registro.
- Permitir que el cliente y el servidor verifiquen que su par haya calculado los mismos parámetros de seguridad y que el protocolo de enlace se haya producido sin manipulación por parte de un atacante.

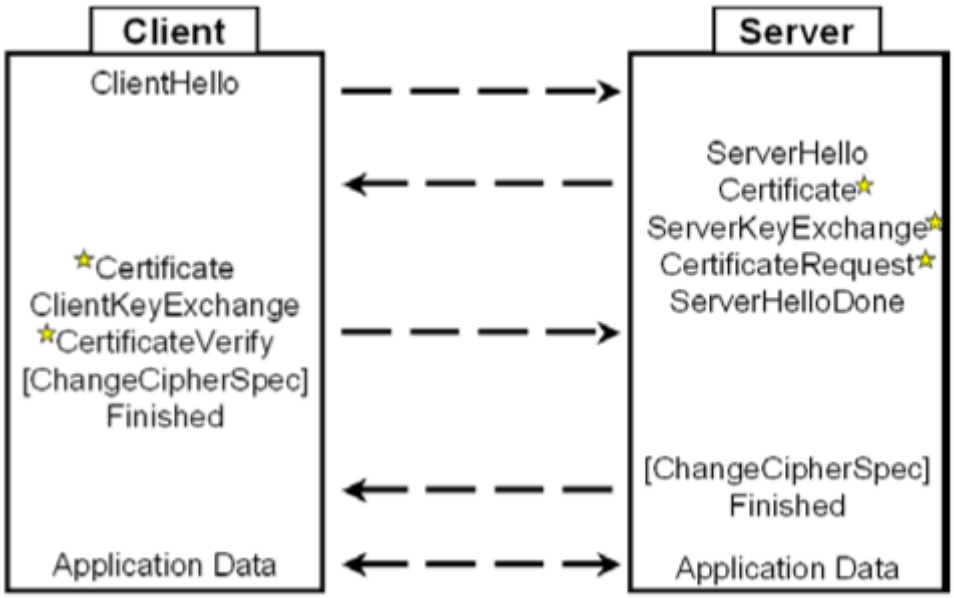

Fig. 6 Diagrama del protocolo TLS HandShake [29].

2.9. BLOCKCHAIN

El concepto de tecnología detrás de *blockchain* es similar al de una base de datos, excepto que la forma en que se interactúa con esa base de datos es distribuida e inmutable, de esta manera los bloques no pueden modificarse, solo añadir nuevos.

Para los desarrolladores, el concepto *blockchain* representa un cambio de paradigma en la forma en que los ingenieros de software escribirán las aplicaciones de software en el futuro, y es uno de los conceptos clave que debe entenderse bien. [30]

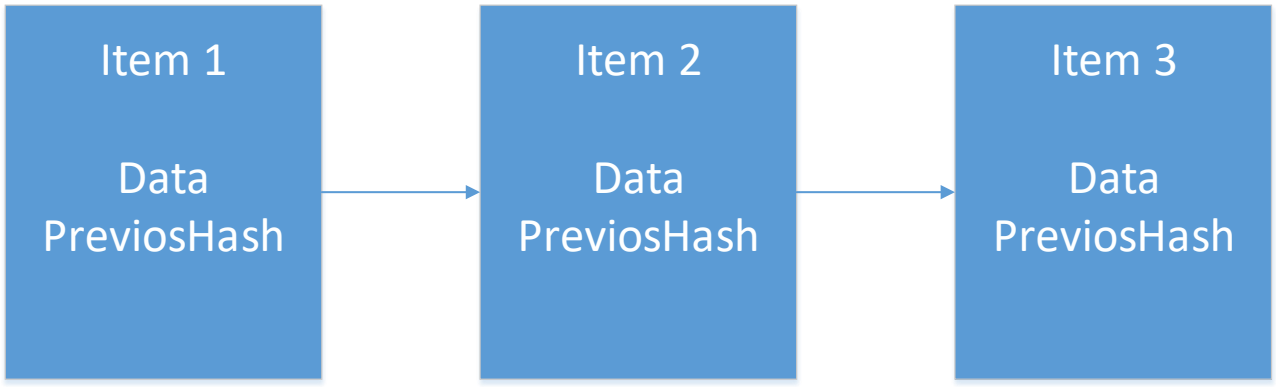

Fig. 7 Diagrama de un blockchain.

Cada eslabón de la cadena contiene la siguiente estructura.

- *Data*: una lista de objetos. Cada elemento de la lista es cada uno de los trozos en los que se ha divido el archivo.
- *Hash*: es una firma digital, que se calcula en función del hash del elemento previo de la lista.
- *TimeStamp*: ofrece información de fecha y hora del momento en el que se añade información de un bloque. [30]

2.9.1. Blockchain publicas

Una cadena de *blockchain* pública, es aquella que cualquiera puede vincularse y participar en las tareas de la red como: leer la cadena, editarla y escribir un nuevo bloque en la cadena. De esta manera, se cumple el principio de un *blockchain* debe ser una herramienta distribuida, descentralizada y autogobernada.

Los ejemplos más conocidos de *blockchains* públicas son *Bitcoin* y *Ethereum*, con las que cualquier persona puede desarrollar una aplicación descentralizada. El cliente puede explotar la red pública ya desarrollada, por un coste variable según el gasto computaciones gracias a *Ethereum*. [31]

2.9.2. Blockchain privadas

Para participar en una red de *blockchain* privada es necesaria una invitación, validada por la persona que inicia la red o por unas reglas preestablecidas. [31]

Los mayores consumidores de *blockchain* privado son las empresas que desean tener una red distribuida internamente o con sus proveedores. A través de una red privada, las empresas pueden aplicar restricciones sobre los usuarios de la red y os acuerdos que se realizan. Se pueden gestionar mediante los participantes de la red o con una autoridad reguladora. [31]

Las plataformas de gestión empresarial sirven para conectar diferentes sistemas y departamentos dentro de una empresa. Dentro de las empresas o en un conjunto de ellas, *blockchain* puede actuar como mediador. [31]

2.9.3. Los servicio blockchain

Los datos se almacenan de forma semipública en una cadena de bloques, de esta manera, cualquiera puede verificar que ha introducido la información porque el contenedor tiene su firma, pero sólo el usuario puede desbloquear lo que hay dentro del contendor, ya que él es el único que tiene la clave privada. [30]

Por lo tanto, la cadena de bloques se comporta como una base de datos, pero parte de la información que está almacenada, su encabezado, es pública. [30]

Los datos almacenados pueden ser un símbolo de valor o un saldo de dinero criptográfico. Entonces, el *blockchain* actúa como un sistema de transferencia de valor alternativo que ninguna autoridad central o tercera parte potencialmente maliciosa puede alterar, debido a que está cifrado. [30]

2.9.4. Computación de confianza

Cuando combina los conceptos detrás del *blockchain*, el consenso descentralizado y los contratos inteligentes, comienza a darse cuenta de que permiten la distribución de recursos y transacciones lateralmente de una manera plana, de igual a igual, y, al hacer eso, están permitiendo que las computadoras confíen unas en otras a un nivel profundo. [30]

Mientras que las instituciones y las organizaciones centrales eran necesarias como autoridades de confianza, un cierto número de sus funciones centrales se pueden codificar a través de contratos inteligentes que se rigen por un consenso descentralizado sobre un *blockchain*. [30]

2.9.5. Aplicaciones

	USUARIO DE PROTOCOLO	FRECUENCIA	BENEFICIOS	EJEMPLOS
MONEDA	Intercambios, procesadores de pagos, carteras	Esporádico	Coste y velocidad	Coinbase, ChangeTip, any wallet, any exchange
SERVICIOS VINCULADOS	Negocio Web	Crónico	Apertura, flexibilidad, nuevos modelos de negocios, efectos de red, usuarios empoderados.	OneName, Mine, Swarm, Streamium, OpenBazaar, Assembly
CONTRATOS INTELIGENTES	Proveedor de servicios de contratos, aplicaciones web o usuario final con herramientas de autoservicio.	Episódico	Autonomía, costo, velocidad.	Mist (by Ethereum), SmartContract, Secure Asset Exchange
DECENTRALIZED AUTONOMOUS ORGANIZATIONS	DAO	Habitual	Protección del usuario, voz del usuario, gobierno del usuario, transparencia, autorregulación, soberanía.	La'Zooz, Storj, MaidSafe, OpenGarden, Bitnation

Tabla IV Aplicaciones de blockchain [30].

2.10. IPFS

El sistema de archivos interplanetario (*InterPlanetary File System*, IPFS) es un sistema de archivos distribuido de *peer-to-peer* que busca conectar todos los dispositivos con el mismo sistema de archivos. IPFS es similar a la Web, pero IPFS podría verse como un único *Cloud* de *BitTorrent*, intercambiando objetos dentro de un repositorio Git. En otras palabras, IPFS proporciona un modelo de almacenamiento en bloque con dirección de contenido de alto rendimiento. IPFS combina una tabla hash distribuida, un intercambio de bloques incentivado y un espacio de nombres de identificación selectiva. IPFS no tiene un único punto de fallo y los nodos no necesitan confiar en los demás. [32]

3. DISEÑO

En este capítulo se explica el diseño y organización del sistema de almacenamiento *Fog*.

Lo primero que tenemos que hacer la virtualización de máquinas, de distintas IPs, de cara a simular *edges* de la red. Cada *edge* actuará como cliente y servidor simultáneamente. De cara a facilitar el desarrollo, se ha trabajado con una máquina como cliente y el resto servidores.

Para la virtualización de las máquinas se ha usado la herramienta Mininet, explicada previamente.

Las funciones disponibles para el cliente son:

1. Envío de un archivo al servidor
2. Recibir un archivo del servidor
3. Servidores disponibles
4. Listar los archivos compartidos
5. Eliminar un archivo
6. Salir

3.1. ENVÍO DE UN ARCHIVO AL SERVIDOR

El cliente tiene la opción de enviar un archivo al servidor, para ello debe elegir la opción 1 del menú principal. Posteriormente debe elegir a dónde quiere enviar el archivo: si a un servidor concreto, a todos o dividir la información entre los servidores disponibles, teniendo en cuenta si los servidores están llenos.

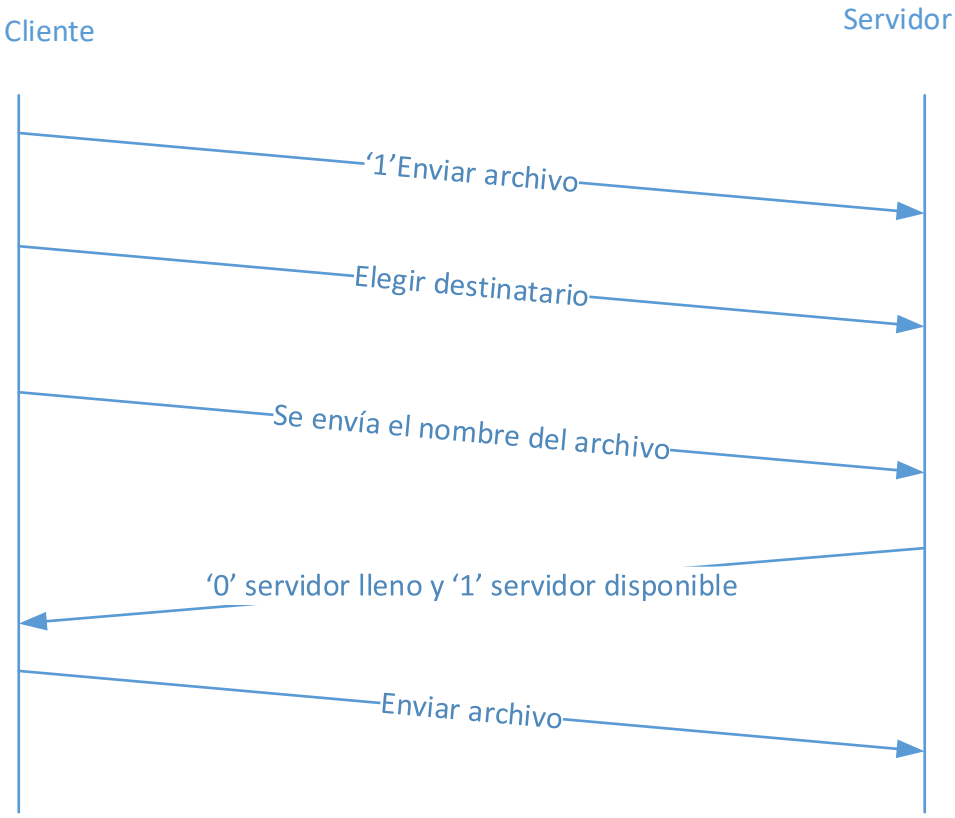

Fig. 8: Diagrama de envío de archivos.

Por último, el usuario debe elegir que archivo desea enviar, escribiendo el nombre con la extensión a enviar.

3.2. RECIBIR UN ARCHIVO DEL SERVIDOR

El cliente tiene la opción de recibir un archivo de un servidor en la opción 2 del menú principal. Para ello debe elegir desde dónde desea recibir el archivo de un servidor concreto o un archivo que ha sido dividido previamente entre distintos servidores.

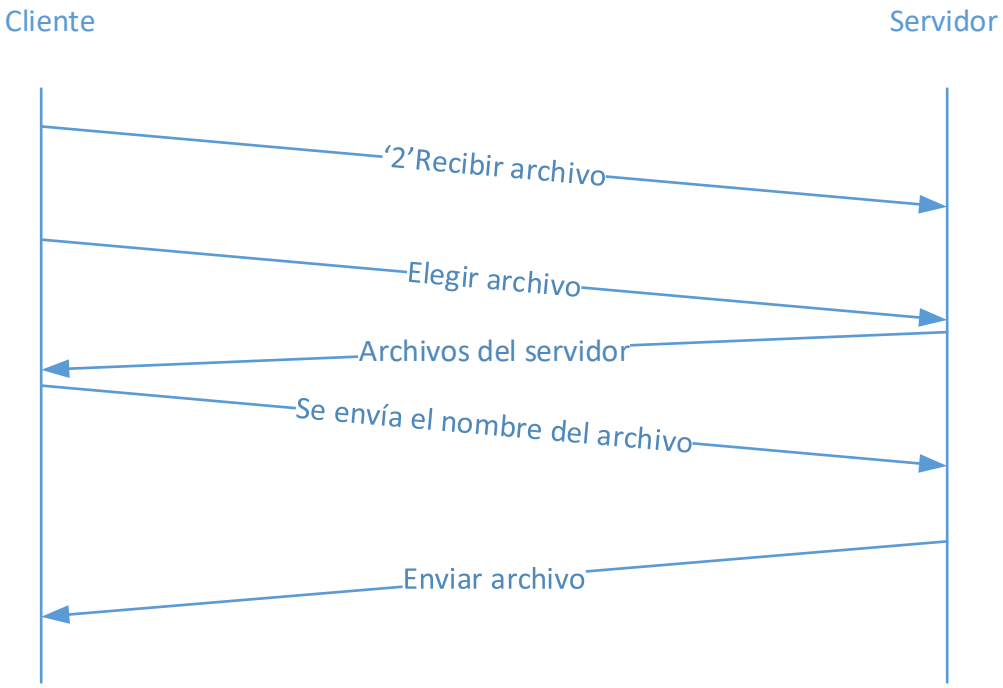

Fig. 9 Diagrama recepción de archivo.

Si se elige recibir de un archivo concreto, el usuario recibe un listado con todos los archivos que contiene el servidor elegido y solo ha de escribir el nombre. Sin embargo, si se trata de un archivo compartido, listamos los archivos que han sido divididos entre los distintos servidores, el usuario elige el archivo y se recupera los fragmentos en los que el archivo ha sido divido entre los distintos servidores.

3.3. SERVIDORES DISPONIBLES

Esta función permite conocer el estado de cada uno de los servidores, recibiendo de cada uno de los servidores conectados, su IP, los bytes de los archivos que contiene y lo bytes disponibles. El tamaño máximo de cada servidor se ha añadido como una nueva variable, inicializada a 5 Gb.

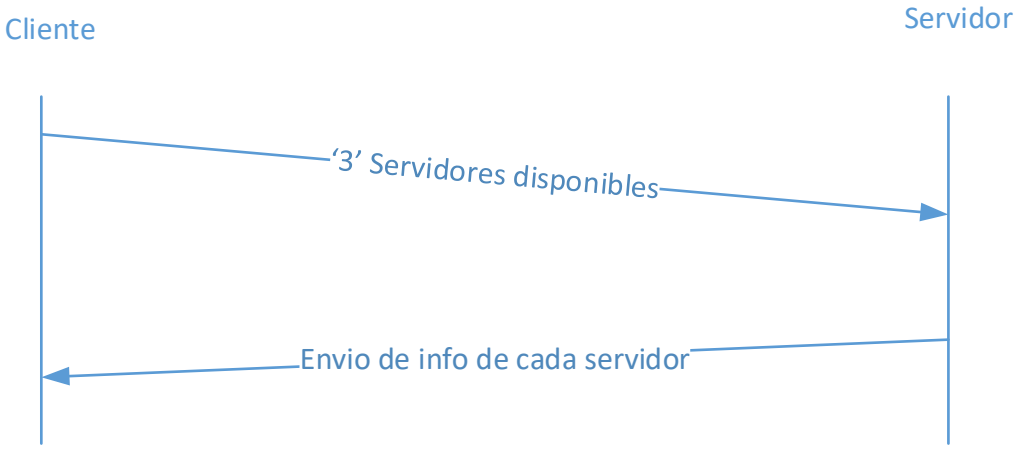

Fig. 10 Diagrama de servidores disponibles.

3.4. LISTAR LOS ARCHIVOS COMPARTIDOS

Saber qué archivos han sido compartidos en cada momento y el historial que han seguido es importante, es por ello por lo que se ha desarrollado la funcionalidad que lista los archivos contenidos en el *blockchain*.

3.5. ELIMINAR UN ARCHIVO

Al igual que se añaden archivos a los distintos servidores, es también importante dar la posibilidad de eliminar los archivos de la carpeta del cliente y de cada uno de los servidores en los que se ha compartido, consultado a la cadena *blockchain*.

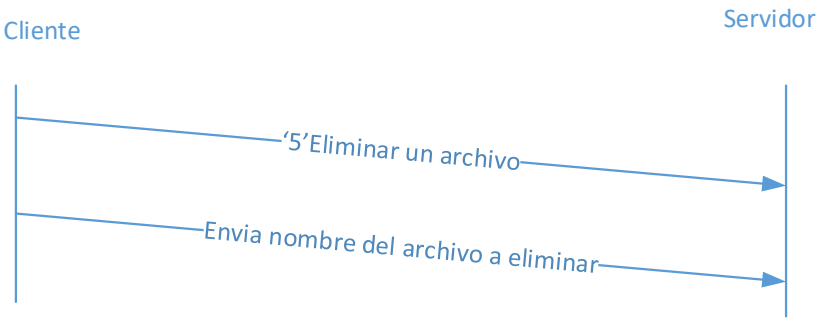

Fig. 11 Diagrama de borrado de archivo.

3.6. SALIR

Para cerrar los sockets de forma correcta, se genera una opción de cierre de sockets.

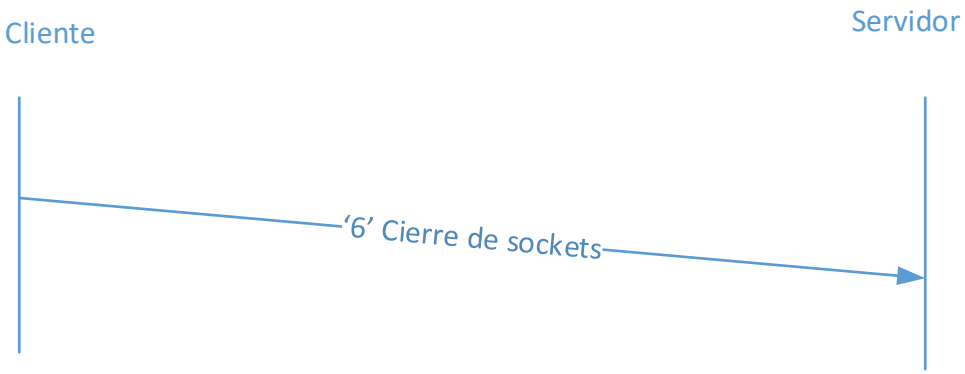

Fig. 12 Diagrama de cierre de sockets.

4. DESARROLLO

El desarrollo de esta aplicación se ha llevado a cabo mediante un programa desarrollado en Java. Se ha utilizado una máquina virtual de Linux para poder usar por terminal el programa Mininet.

Ejecutamos el comando explicado previamente en el apartado 3 de cara a simular un cliente y tres servidores. Cada una de las máquinas tiene su propia IP y además, tiene su propio fichero, el cual contiene el .java y .class.

4.1. ESTABLECER CONEXIÓN CLIENTE/SERVIDOR

El inicio del código consiste es la creación de sockets entre cliente y servidor para un intercambio básico de información.

El cliente utiliza socket de tipo Socket, indicando con qué dirección IP se quiere establecer la conexión y el puerto por el que se envían los datos

```
Socket(InetAddress address, int port)
```

Y el servidor está escuchando mediante la inicialización de un socket de tipo serverSocket, indicando el puerto en el que se escucha.

```
ServerSocket(int port).
```

Se han definido un objeto de tipo Address, que contiene la IP y el puerto. Para trabajar con los servidores se ha creado una lista de tipo Address, que se utilizará para recorrer todos los servidores conectados.

De esta manera, recorremos la lista de servidores, estableciendo las conexiones con cada uno de los servidores.

Una vez se ha establecido una conexión básica, procedemos a utilizar el protocolo TLS, para lo que hay que crear un certificado

Para Crear el certificado hay que seguir los siguientes pasos. [33]

$ keytool -genkey -alias diegoKeystore -keystore diegoKeystore

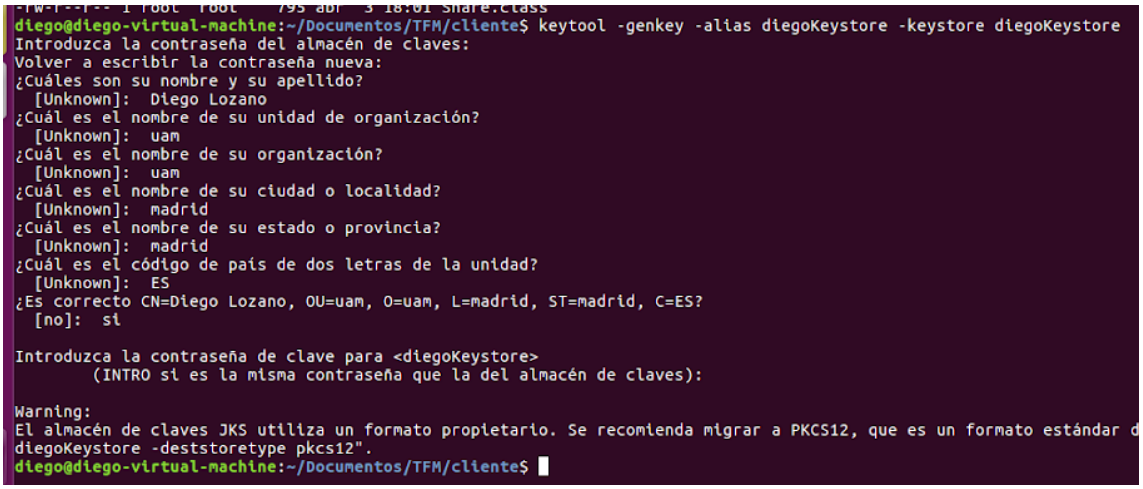

Fig. 13 Pasos para la creación del certificado.

Y se rellenan los campos según van apareciendo en la terminal.

Una vez creado se comprueba mediante el comando *seahorse* en la terminal de Linux.

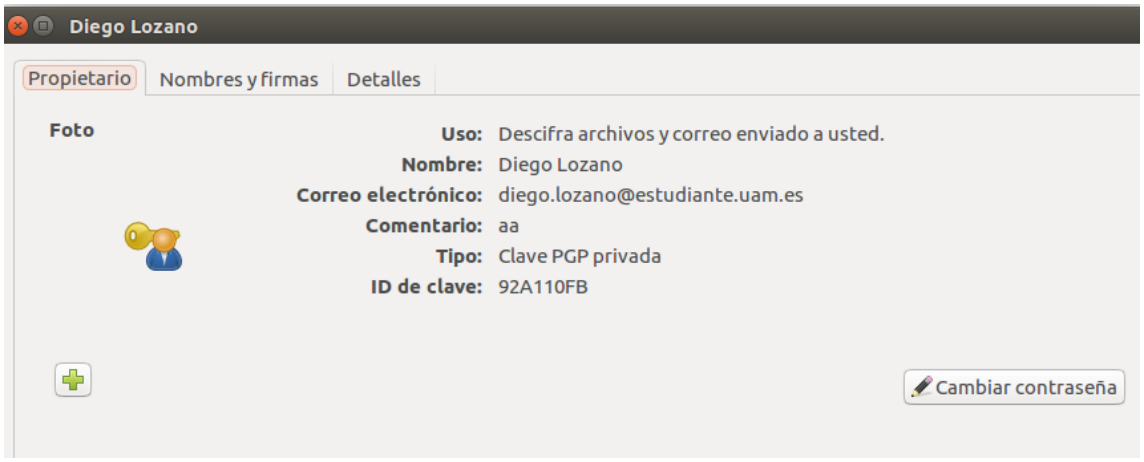

Fig. 14 Certificado.

Para utilizar el protocolo SSL, se ha de cambiar el tipo de socket que vamos a usar. En el caso del cliente, se pasa a usar SSLSocketFactory

```
SSLSocketFactory sslSocketFactory = (SSLSocketFactory)SSLSocketFactory.getDefault();

miSocket=sslSocketFactory.createSocket(InetAddress address, int port)
```

Y en el servidor se pasa a usar SSLServerSocketFactory.

```
SSLServerSocketFactory sslServerSocketFactory=(SSLServerSocketFactory)SSLServerSocketFactory.getDefault();

servidor = sslServerSocketFactory.createServerSocket(9999);
```

4.2. INTERCAMBIO DE COMANDOS ENTRE CLIENTE Y SERVIDOR

Para saber en qué punto está el cliente y el servidor, se intercambian una serie de comandos para indicar que desea hacer el usuario.

Para el envío desde cliente a servidor de un comando se utiliza:

```
PrintWriter out = new PrintWriter(listaSockets.get(h).getOutputStream(), true);
```

Capturamos el comando por teclado mediante:

```
BufferedReader bufferedReader =new BufferedReader(new InputStreamReader(listaSockets.get(h).getInputStream()));
Scanner scanner = new Scanner(System.in);
```

Y por último enviamos el comando:

```
out.println(opcionMenu);
```

En el lado del servidor, escuchamos la información enviada por el cliente.

```
PrintWriter out = new PrintWriter(misocket.getOutputStream(), false);
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(misocket.getInputStream()));
String line= null;
while((line = bufferedReader.readLine()) != null){
        out.println(line);
        break;
}
```

En caso de que se quiera enviar comandos del servidor al cliente el mecanismo será el mismo.

4.3. MENÚ PRINCIPAL

El menú principal cuenta con las opciones indicadas en el capítulo anterior. Para que sea más interactivo, se ejecutan todas las opciones dentro de un *while loop*, de manera que después de hacer una acción se pueda continuar con otra. Después de cada interacción es necesario inicializar de nuevo algunas variables.

La primera opción que debe elegir el usuario es elegir que opción desea hacer. En función de la opción, mediante un *if-else.*

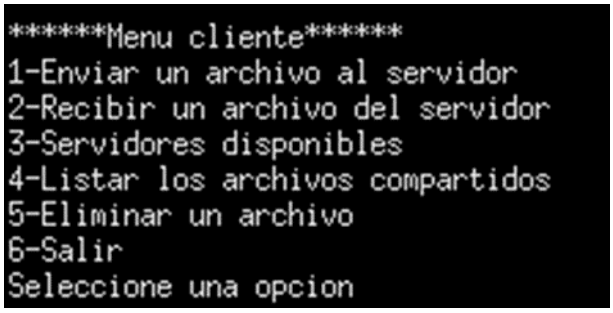

Fig. 15 Menú principal.

4.4. ENVÍO DE ARCHIVOS

El envío del cliente y servidor se ha externalizado en la función enviar y enviarVarios.

El método enviar tiene como parámetros de entrada:

- String file: contiene la ruta absoluta del archivo que se quiere enviar.
- Socket miSocket: El socket de destino al cual se desea enviar el archivo.
- Cliente ske: que contiene la password para cifrar el contigo mediante el algoritmo AES.

Lo primero que se realiza es una lectura del archivo a enviar, almacenando la información en un objeto de tipo *BufferedInputStream*. Posteriormente se escribe la información leída del archivo en el socket, utilizando los métodos *getOutputStream*() de la clase Socket y *OutputStream*. Por último, mediante el método *flush*, se vacía la secuencia de bytes almacenadas en el objeto de tipo *OutputStream*.

También se ha desarrollado la opción de enviar un archivo a todos los servidores, para ellos, se invoca al método enviar, descrito en esta sección, tantas veces como servidores haya en la lista, con un *for loop*, variando el servidor al que se lo queremos enviar.

Las pantallas por las que va pasando el usuario son las siguientes:

```
Seleccione a donde desea enviarlo
1-Enviarlo a2e0fa5d3[SSL_NULL_WITH_NULL_NULL: Socket[addr=/10.0.0.2,port=9999,localport=51836]]
2-Enviarlo a5010be6[SSL_NULL_WITH_NULL_NULL: Socket[addr=/10.0.0.3,port=9999,localport=60526]]
3-Enviarlo a685f4c2e[SSL_NULL_WITH_NULL_NULL: Socket[addr=/10.0.0.4,port=9999,localport=48714]]
4-Dividir
5-Enviar a todos
```

Fig. 16 Envío de un archivo: elegir destinatario.

```
1-ccc.pdf
2-Cliente.class
3-Address.class
4-250.txt
5-Cliente.java
6-500.txt
7-.goutputstream-FCC2KZ
8-ww.jpg
9-blockchain
10-10.txt
11-50.txt
12-Share.class
13-bs
14-1.txt
15-250MB.txt
16-1000MB.txt
17-inn
Escriba el archivo a enviar
```

Fig. 17 Envío de un archivo: listado de archivos.

```
Accepted connection : 2e0fa5d3[TLS_DHE_DSS_WITH_AES_256_CBC_SHA256: Socket[addr=/10.0.0.2,port=9999,localport=51848]]
Sending /home/diego/Documentos/TFM/cliente/1.txt(1000016 bytes)
Done.
---------Finished Script---------
```

Fig. 18 Envío de un archivo: confirmación de envío.

4.4.1. Envío a varios

Esta función se utiliza en caso de que se quiera dividir un archivo entre varios servidores. Para ello lo primero que hay que hacer es calcular el número de servidores disponibles. En función de los archivos disponibles, se calcula el número de trozos a dividir la información y el número de bytes que va en cada servidor. En caso de un servidor este lleno, se hace un recálculo sin utilizar ese servidor, y se divide entre los servidores disponibles. En caso de que el número de bytes no sea divisible entre el número de servidores, el ultimo servidor de la lista contendrá los bytes sobrantes, evitando así posibles errores.

Una vez se ha determinado el número de archivos en los que se ha dividido, se procede a enviar los archivos invocando al método enviar, descrito en este apartado.

4.5. RECIBIR UN ARCHIVO

Para la recepción de un archivo, se ha implementado un método llamado recibir, que tiene como parámetros de entrada:

- Socket miSocket: El socket que hay abierto con el origen del archivo que se va a recibir.
- String file: contiene el nombre del archivo que se quiere recibir.
- Cliente ske: que contiene la password para descifrar el contigo mediante el algoritmo AES.

Lo primero que se hace es realizar un barrido, leyendo los bytes que se lean del objeto *InputStream*. Cuando el número de bytes leídos sea nulo, se sabrá que se ha llegado al final del archivo, almacenando los bytes en un *array*. Una vez se ha terminado de leer, se procede a escribir en un archivo nuevo creado. Por último, mediante un *flush* se vacía el objeto de tipo *BufferedOutputStream*.

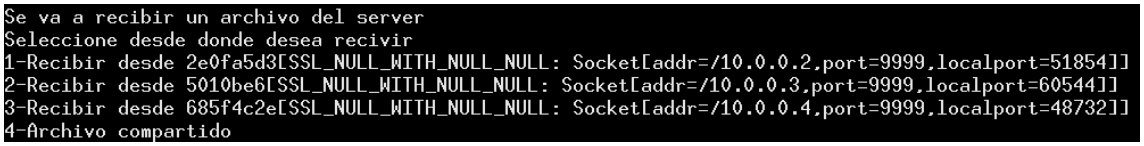

Fig. 19 Recibir un archivo: elegir el servidor de origen.

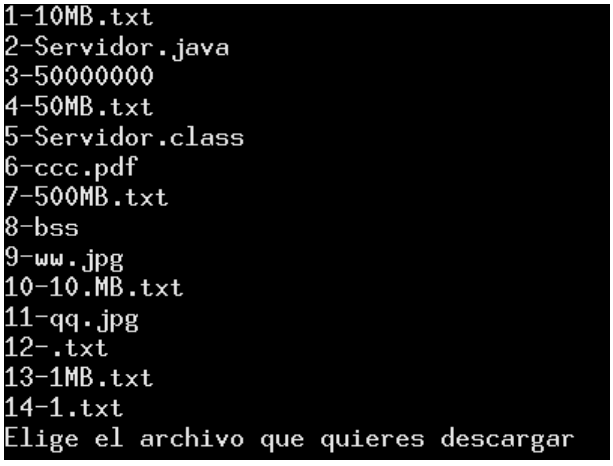

Fig. 20 Recibir un archivo: listado de archivos del servidor.

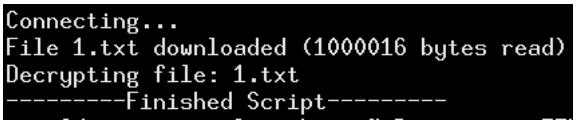

Fig. 21 Recibir un archivo: confirmación de recepción.

4.6. SERVIDORES DISPONIBLES

Para saber el estado de cada uno de los servidores, el usuario debe elegir la opción 3 del menú principal. Este comando se envía a todos los servidores, por lo que utilizamos un for loop recorriendo la lista de servidores conectados.

El servidor para calcular su capacidad, obtiene todos los archivos de su directorio y va sumando el tamaño de cada archivo, enviando ese valor al cliente. La capacidad máxima de cada servidor se ha establecido por defecto 5 GB.

Cuando el cliente recibe la información de cada servidor, la recibe en bytes. De cara a una mejor visualización se ha calculado la capacidad de cada servidor en distintas unidades, bytes, kB, MB o GB según la unidad que mejor se adapte.

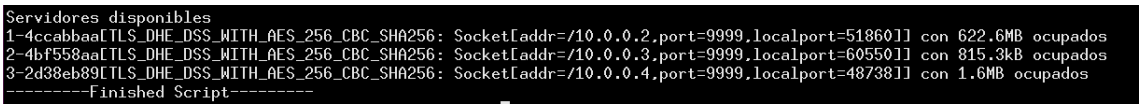

Fig. 22 Estado de los servidores.

4.6.1. ELIMINAR UN ARCHIVO

La opción 5 del menú elimina un archivo: para ello se listan todos los archivos que tiene el cliente en la carpeta local y el usuario debe elegir el archivo a eliminar. El nombre del archivo se elimina a todos los servidores y se elimina en caso de que tengan ese archivo. Además, se elimina de la carpeta local del cliente

4.7. ENCRIPTADO DE ARCHIVOS

El objetivo de enviar la información encriptada es que, en caso de que el servidor sufra un ataque, nuestra información no sea pública. Al cifrar el contenido en origen evitamos enviar la información directamente. Para cifrar se ha utilizado una clave simétrica como AES.

En el método de envío de un archivo del cliente al servidor se crea un archivo encriptado que será el que luego se envía al servidor. Una vez enviado este archivo es borrado. De esta manera, se envía el encriptado y el original se queda en el cliente.

En la recepción de un archivo desde el servidor, se recibe el archivo cifrado, se descifra y se escribe en un archivo nuevo, eliminando el archivo encriptado recibido.

4.8. MÉTODOS AUXILIARES

- *applySha256* (*String* input): este método es utilizado para calcular el Hash en cada bloque del *blockchain*. Como parámetros de entrada tenemos el hash del bloque anterior y el *TimeStamp*.
- *calculateHash*: concatena la cadena entera del *hash* de cada bloque del *blockchain*.
- Tiempo: Es utilizado para calcular el tiempo que tarda el programa en hacer cada acción, utilizado para realizar pruebas.
- *encryptFile* (*File* f): se usa este método para la encriptación del archivo antes de enviarlo. Como parámetro de entrada se pasa el archivo a cifrar.
- *decryptFile* (*File* f): se usa este método para la descifrar el archivo cuando se ha recibido. Como parámetro de entrada se pasa el archivo a descifrar.
- *writeToFileSend* (*File* f): método auxiliar del método *encryptFile*, que crea el archivo encriptado en función del archivo original. Como parámetro de entrada se pasa el archivo a cifrar.
- *writeToFileReceive* (File f): método auxiliar del método *decryptFile*, que crea el archivo original, a partir del archivo encriptado. Como parámetro de entrada se pasa el archivo a descifrar.

4.9. BLOCKCHAIN

La cadena *blockchain* se ha creado a partir de la siguiente estructura, de tipo share:

Fig. 23 Ejemplo de blockchain.

- Dirección IP: IP de la máquina donde se ha enviado ese fragmento de archivo
- Nombre: nombre del archivo completo
- Nombre del fragmento: Nombre del fragmento que se ha enviado al servidor.
- TimeStamp: Fecha y hora de la creación del bloque

Los archivos compartidos se añaden al *blockchain.*

```
Listamos los archivos compartidos
Archivo ccc.pdf la parte ccc.pdf.part1 esta en /10.0.0.2
Archivo ccc.pdf la parte ccc.pdf.part2 esta en /10.0.0.3
Archivo ccc.pdf la parte ccc.pdf.part3 esta en /10.0.0.4
Archivo ww.jpg la parte ww.jpg.part1 esta en /10.0.0.2
Archivo ww.jpg la parte ww.jpg.part2 esta en /10.0.0.3
Archivo ww.jpg la parte ww.jpg.part3 esta en /10.0.0.4
```

Fig. 24 Archivos compartidos.

4.9.1. Serialización del blockchain

El objetivo del *blockchain* es mantener el historial de cada uno de los bloques de los archivos. Para no perder el histórico cada vez que se ejecuta el programa, cada vez que se sale del programa se realiza una serialización de la cadena de archivos. Al arrancar el programa se comprueba si existe una cadena de archivos ya creada, en ese caso, se carga el *blockchain* ya existente, sino se creara una nueva cadena.

5. INTEGRACIÓN, PRUEBAS Y RESULTADOS

En este capítulo se explican las pruebas llevadas a cabo con el fin de validar el software y analizar qué mejoras que ofrece con respecto a otras tecnologías. Para ello se ha hecho un plan de pruebas en distintos contextos para comparar los resultados.

Primero, se analizará el tiempo que se tarda en realizar diferentes operaciones con el sistema desarrollado. Para realizar las pruebas de tiempos, hemos realizado un script *bash*, de manera que se ejecute *n* veces, para que los tiempos medidos sean significativos. Para pasar los parámetros que debe ir eligiendo el usuario, se ha modificado el código, pasando estos valores como argumentos:

1- Opción del menú principal.
2- Servidor al que se desea enviar.
3- Archivo a intercambiar.

Para ello, se ha repetido cada prueba 30 veces, calculando la media y la desviación típica. Se ha utilizado la herramienta *Matlab*, para su representación con el comando *errorbar*.

5.1. ENTORNO DE PRUEBAS

Para crear nuestra red se ha utilizado el siguiente comando en una terminal de Linux.

```
sudo mn --topo linear,4
```

En este caso se han simulado cuatro máquinas: una que actuará como cliente y las otras 3 actuarán como servidores.

IP Cliente: 10.0.0.1

IP Servidor: 10.0.0.2

IP Servidor1: 10.0.0.3

IP Servidor2: 10.0.0.4

Se pueden simular tantas máquinas como se desee, pero para el desarrollo y realización de pruebas se ha optado por este escenario.

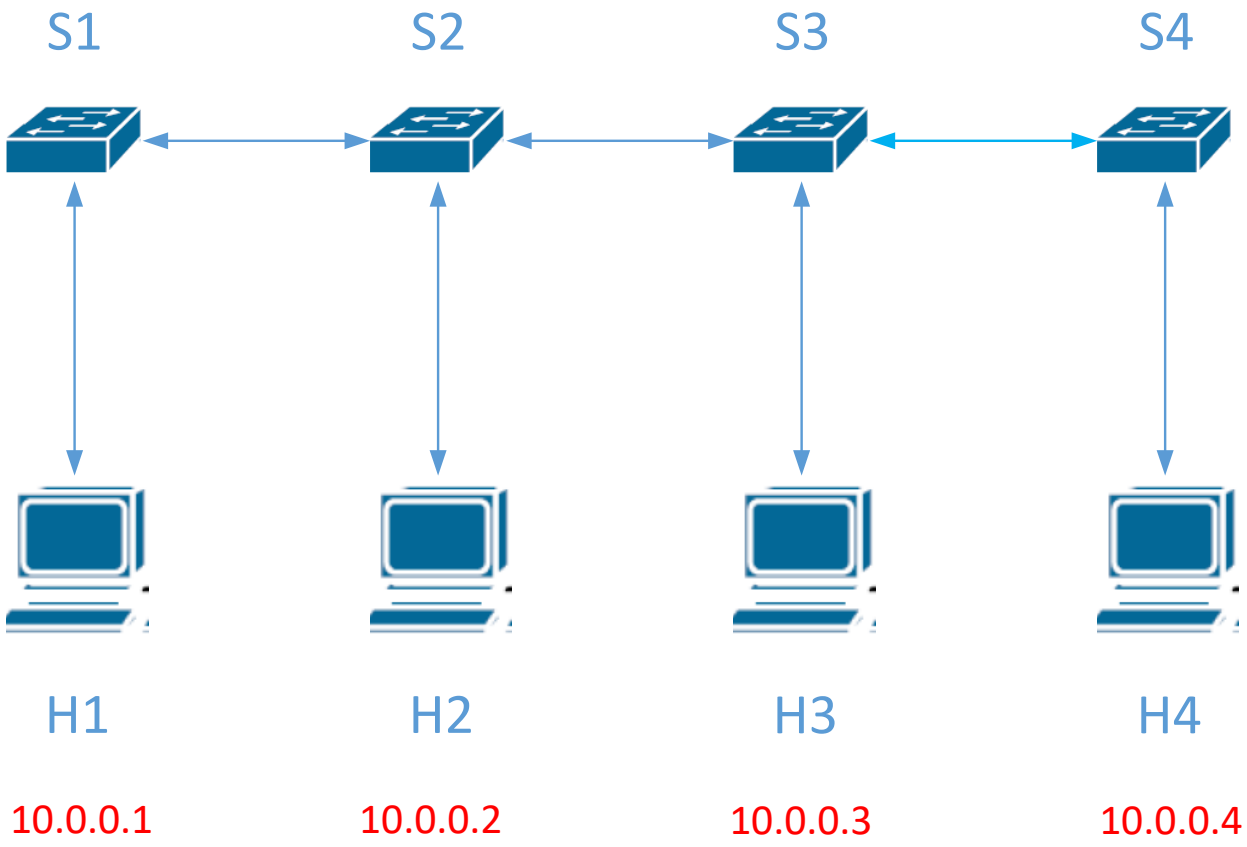

Fig. 25 Red simulada con Mininet.

En la terminal que actúa como cliente se ejecutará el siguiente comando:

```
java -Djavax.net.ssl.trustStore=/home/diego/Documentos/TFM/mykeystore/diegoKeystore -
Djavax.net.ssl.trustStorePassword=alumnoeps -Xms1024m -Xmx4096m Cliente
```

En la terminal que actúa como servidor se ejecutará el siguiente comando:

```
java -Djavax.net.ssl.keyStore=/home/diego/Documentos/TFM/mykeystore/diegoKeystore -
Djavax.net.ssl.keyStorePassword=alumnoeps -Xms1024m -Xmx2048m Servidor
```

5.2. ENVIAR UN ARCHIVO A UN SERVIDOR

Contexto: un cliente con un servidor (1:1). Se envía un archivo de varios tamaños

Para hacer esta prueba se ha usado la herramienta Mininet para simular dos terminales, cada una con su correspondiente IP. Una de ellas actuará como cliente y la otra actuará como servidor.

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos de envío y de encriptado.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no, se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

IPOrigen 10.0.0.1

IPDestino 10.0.0.2

Datos de entrada: para esta opción los argumentos que hemos pasado han sido:

1- Opción del menú principal: enviar un archivo (1).
2- Servidor al que se desea enviar: enviar al servidor (1).
3- Archivo a intercambiar: en cada iteración se va variando el archivo.

Resultado esperado:

El objetivo es obtener un archivo con los tiempos de inicio de acción, finalizado de encriptado y finalizado de acción.

El archivo se ha recibido correctamente en destino, con el mismo tamaño y el contenido encriptado.

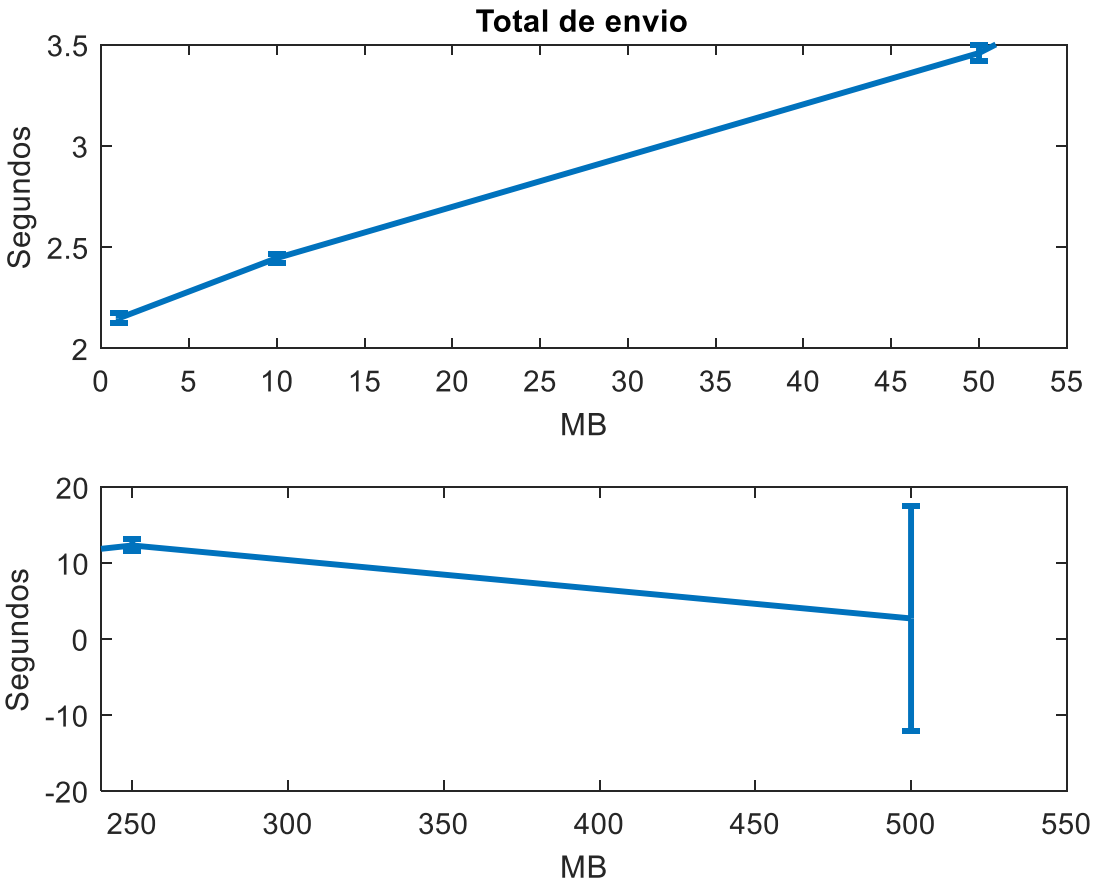

Fig. 26 Comparativa de tiempos de envío y encriptado de un archivo de distintos tamaños.

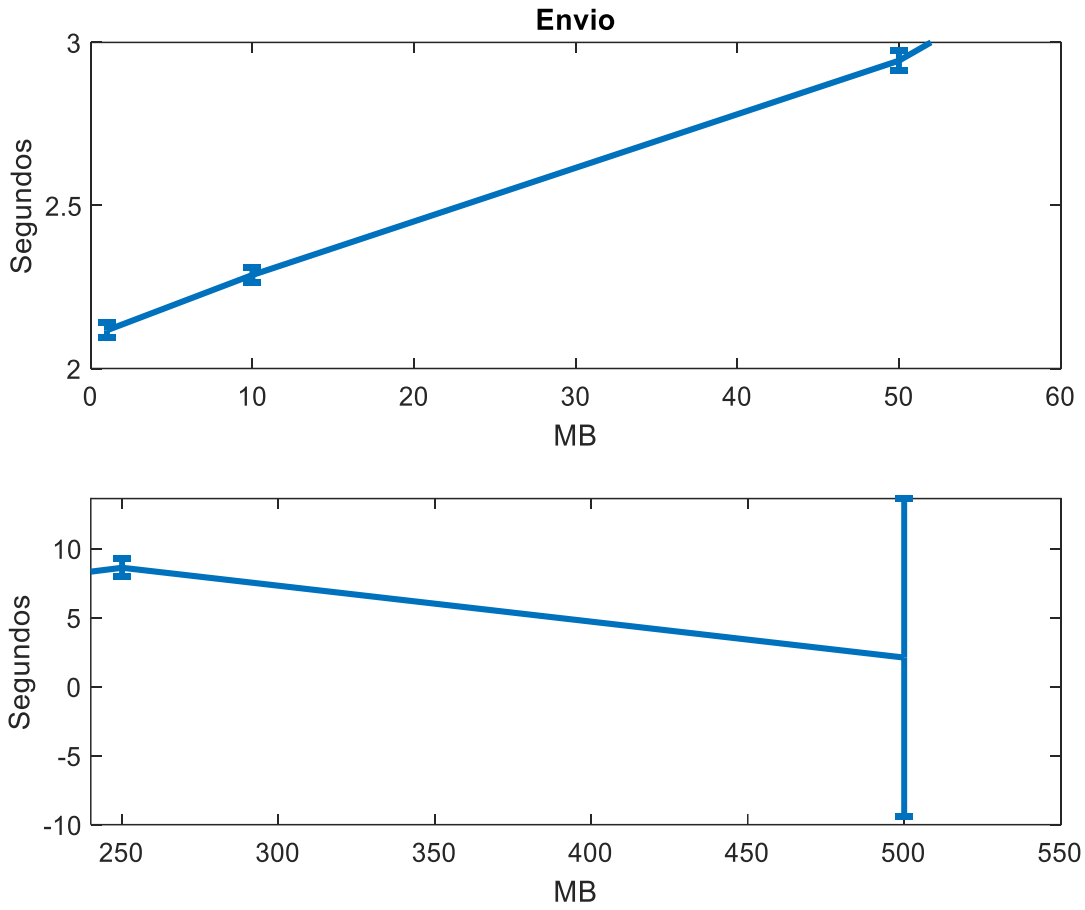

Fig. 27 Comparativa de tiempos de envío de un archivo de distintos tamaños.

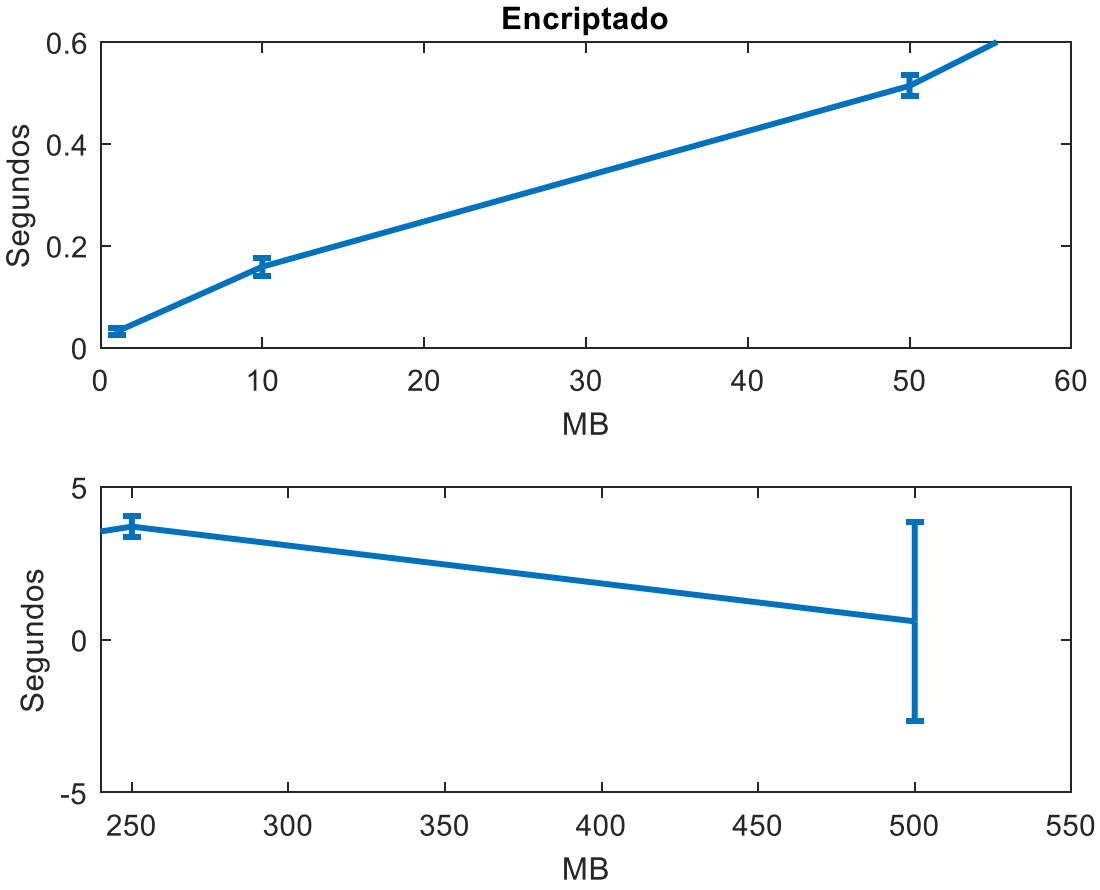

Fig. 28 Comparativa de tiempos de encriptado de un archivo de distintos tamaños.

En este caso los tiempos de envío total de un archivo de 500MB son inferiores al de 250MB debido a la manera de realizar las pruebas. Al enviar archivos de 500MB la prueba requería mucho uso de memoria y no se limpiaba. Las pruebas de archivos de 500MB caben en memoria, mientras que los de 250MB se leen a trozos.

5.3. ENVIAR UN ARCHIVO A VARIOS SERVIDORES

Para hacer esta prueba hemos usado la herramienta Mininet para simular cuatro máquinas virtuales, cada una con su correspondiente IP. Una de ellas actuará como cliente y las otras tres actuarán como servidores.

IPOrigen 10.0.0.1

IPDestino1 10.0.0.2

IPDestino2 10.0.0.3

IPDestino3 10.0.0.4

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos y comprobar su rendimiento.

El archivo se debe fraccionar en tantos trozos como servidores tengamos simulados. En caso de uno de los servidor este lleno, se recalculará para dividir entre los servidores que no estén llenos.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

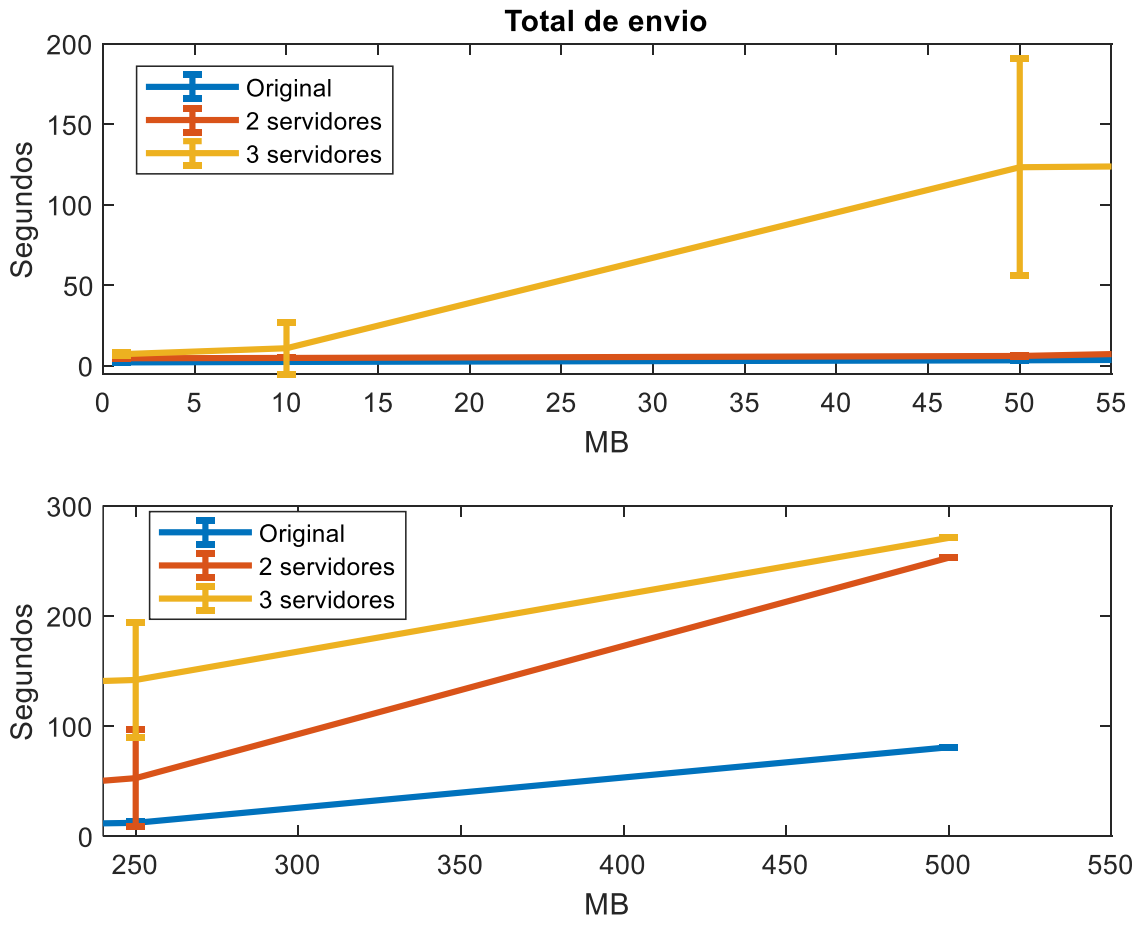

Fig. 29 Comparativa de tiempos de envío y encriptado de un archivo a varios servidores de distintos tamaños.

En este caso observamos que a mayor número de servidores el tiempo de envío total es mayor, debido al mayor número de conexiones que hay que establecer, esto se debe a la forma en la que se ha desarrollado el algoritmo. En nuestro caso se envían los archivos secuencialmente, por lo que el

incremento de tiempo de envío es lógico. Como posible mejora se podría cambiar el algoritmo para enviar en paralelo para optimizar el tiempo.

5.4. ENVIAR UN ARCHIVO A UN ÚNICO SERVIDOR CON PÉRDIDAS DE PAQUETES

Contexto: un cliente con un servidor (1:1). Se envía un archivo de varios tamaños

Para hacer esta prueba se ha usado la herramienta *Mininet* para simular dos terminales, cada una con su correspondiente IP. Una de ellas actuará como cliente y la otra actuará como servidor. Para añadir pérdidas se ha utilizado el comando *loss* de *Mininet*. Se han realizado pruebas para un 2%, 5% y 9 % de pérdidas de paquetes. [34]

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos de envío y de encriptado.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

IPOrigen 10.0.0.1

IPDestino 10.0.0.2

Datos de entrada: para esta opción los argumentos que hemos pasado han sido:

1- Opción del menú principal: enviar un archivo (1).
2- Servidor al que se desea enviar: enviar al servidor (1).
3- Archivo a intercambiar: en cada iteración se va variando el archivo.

Resultado esperado:

El objetivo es obtener un archivo con los tiempos de inicio de acción, finalizado de encriptado y finalizado de acción.

El archivo se ha recibido correctamente en destino, con el mismo tamaño y el contenido encriptado.

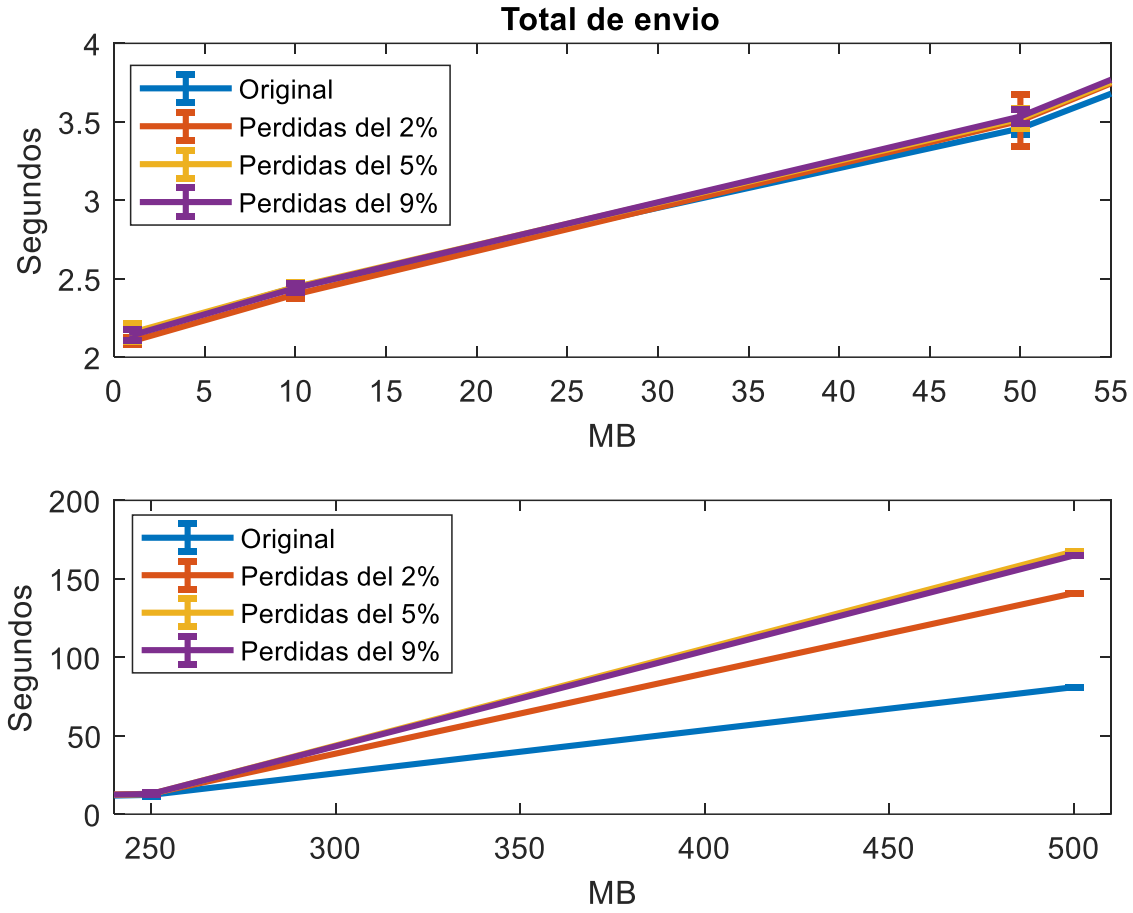

Fig. 30 Comparativa de tiempos de envío y encriptado de un archivo de distintos tamaños con pérdidas.

En este caso se han supuesto distintos escenarios variando las perdidas. En un entorno Wifi con unas pérdidas aproximadas del 2% se observa que el tiempo de transmisión sería de 50 segundos. En el peor de los casos con unas pérdidas del 9% el fichero tardaría 100 segundos. Estos retardos entre los distintos valores de pérdidas son despreciables con archivos de 500MB.

Se aprecia una progresión lineal en función de las pérdidas y del tamaño del archivo, lo que significa cierta coherencia en el resultado y se puede modelar.

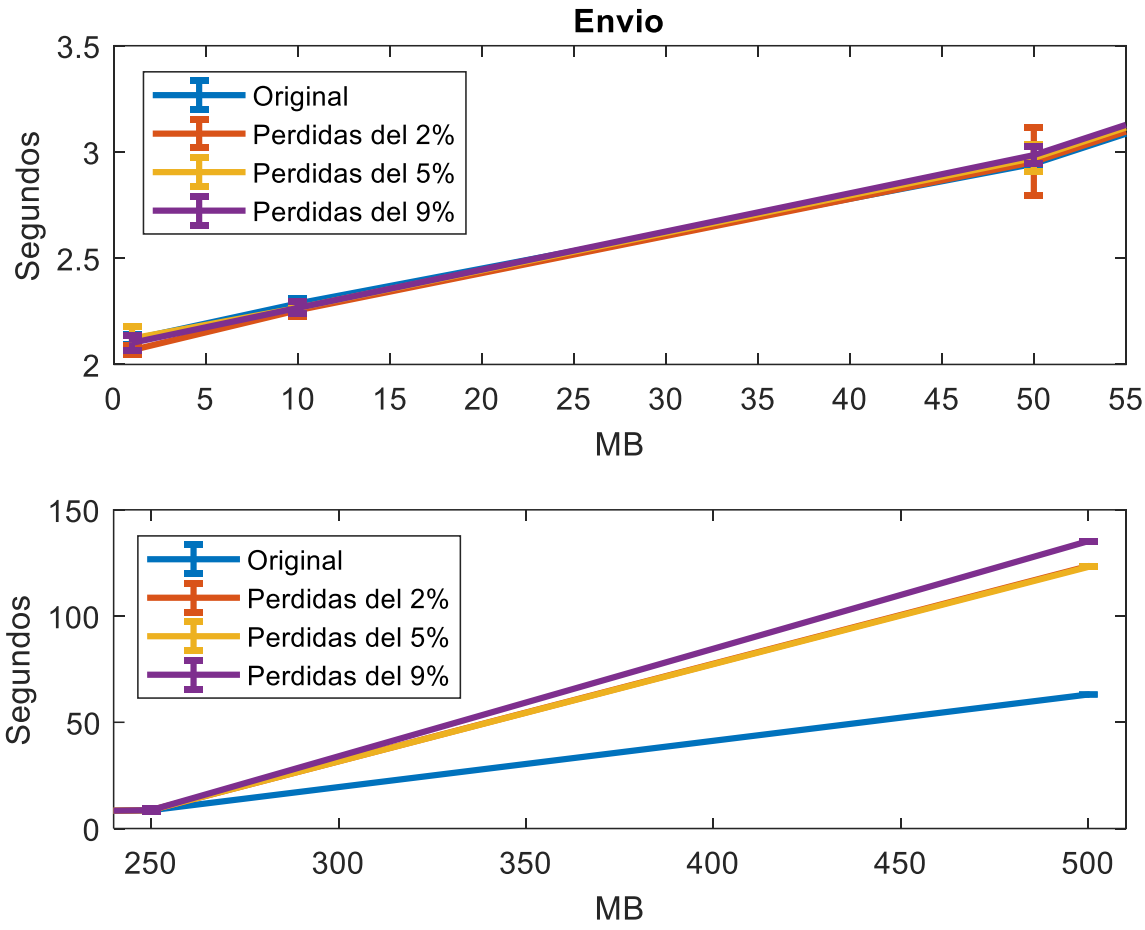

Fig. 31 Comparativa de tiempos de envío de un archivo de distintos tamaños con pérdidas.

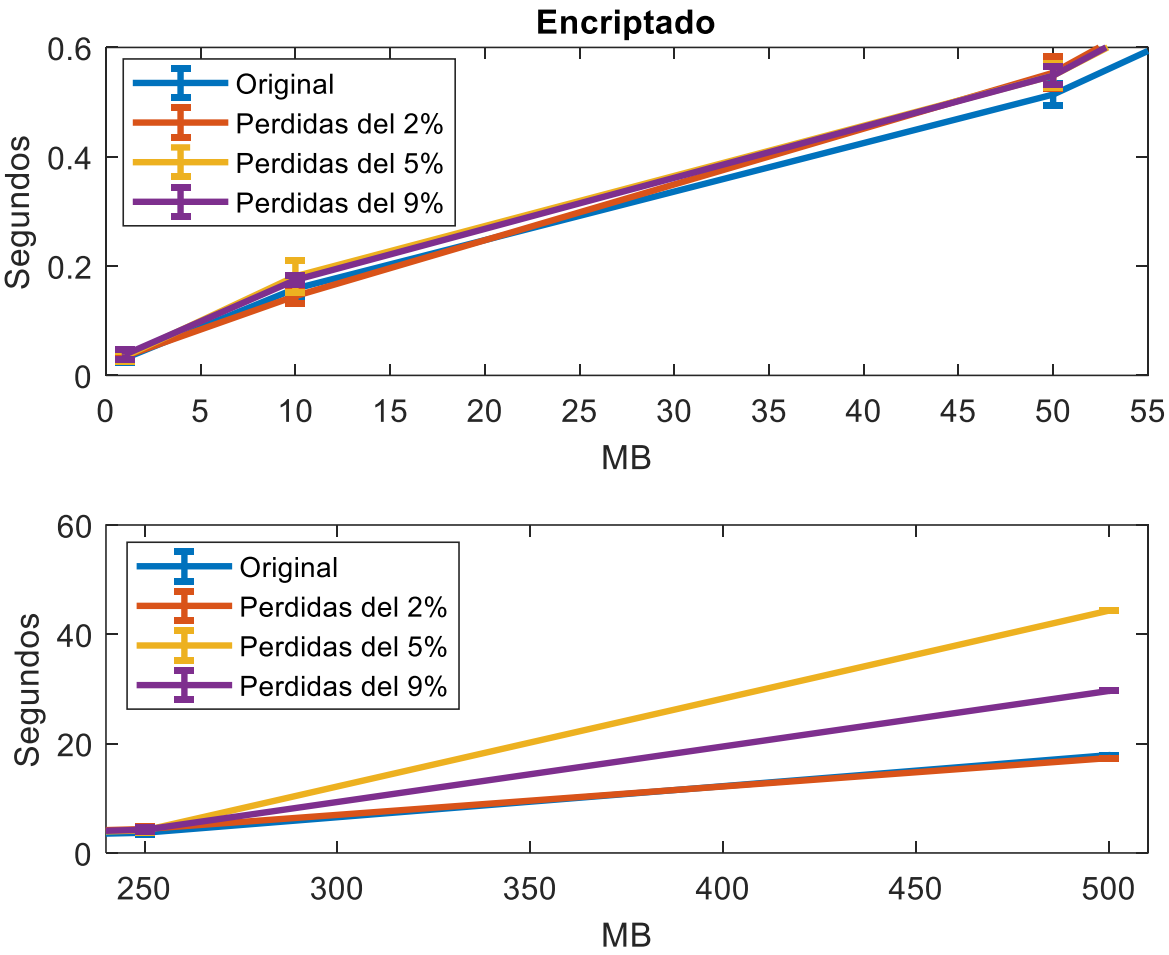

Fig. 32 Comparativa de tiempos de encriptado de un archivo de distintos tamaños con pérdidas.

El tiempo de encriptado no afecta a las perdidas, por lo que no es un factor relevante.

5.5. ENVIAR UN ARCHIVO A UN ÚNICO SERVIDOR CON RETARDOS

Contexto: un cliente con un servidor (1:1). Se envía un archivo de varios tamaños

Para hacer esta prueba se ha usado la herramienta *Mininet* para simular dos terminales, cada una con su correspondiente IP. Una de ellas actuará como cliente y la otra actuará como servidor. Para añadir retardos se ha utilizado el comando *delay* de *Mininet*. Se han realizado pruebas 50ms, 75ms, 100ms y 150ms de retardo, teniendo en cuenta los retardos que se dan dispositivos móviles. [34]

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos de envío y de encriptado.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

IPOrigen 10.0.0.1

IPDestino 10.0.0.2

Datos de entrada: para esta opción los argumentos que hemos pasado han sido:

1- Opción del menú principal: enviar un archivo (1).
2- Servidor al que se desea enviar: enviar al servidor (1).
3- Archivo a intercambiar: en cada iteración se va variando el archivo.

Resultado esperado:

El objetivo es obtener un archivo con los tiempos de inicio de acción, finalizado de encriptado y finalizado de acción.

El archivo se ha recibido correctamente en destino, con el mismo tamaño y el contenido encriptado.

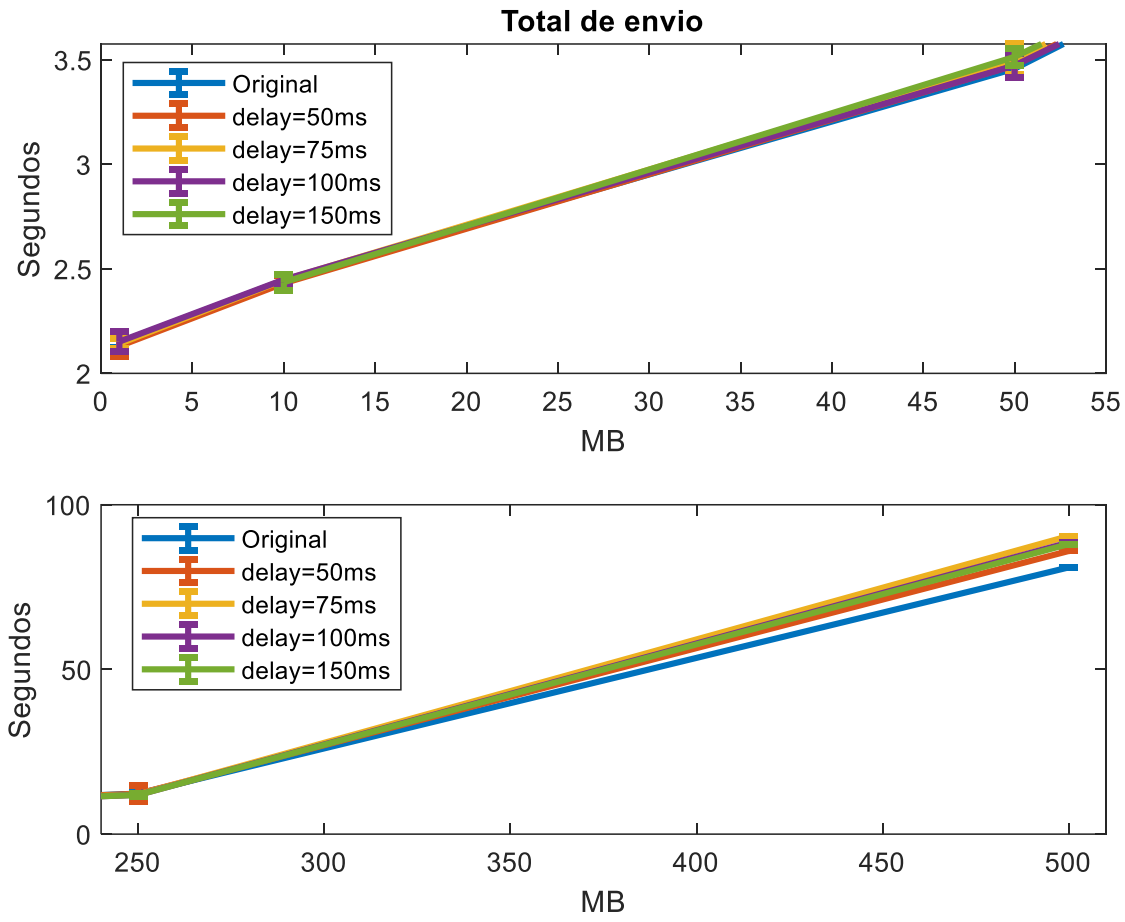

Fig. 33 Comparativa de tiempos de envío y encriptado de un archivo de distintos tamaños con retardos.

En este caso vemos que un retardo de 50-150ms frente a los 90 segundos de envío es despreciable, ya que apenas se nota diferencias entre ellos. Esta prueba se podría reconsiderar introduciendo el retardo en cada uno de los paquetes, debido al retardo que se iría acumulando.

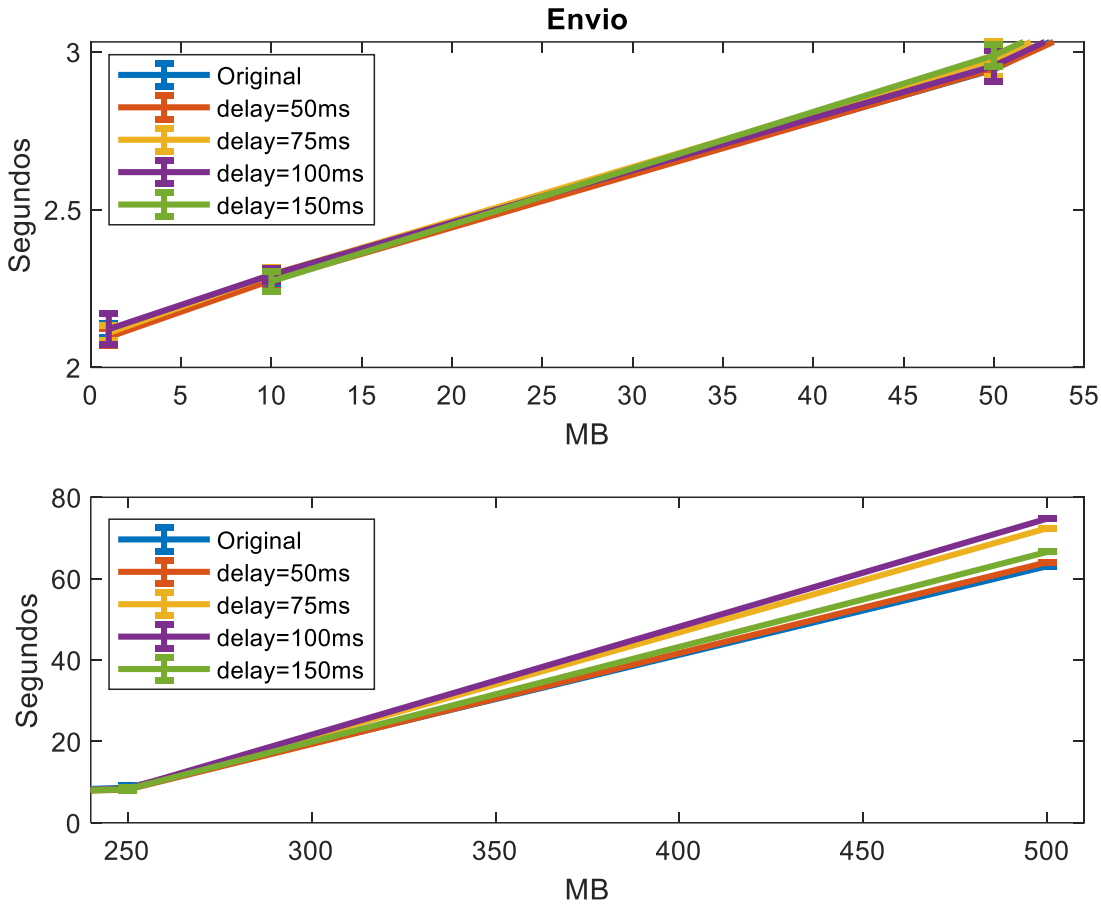

Fig. 34 Comparativa de tiempos de envío de un archivo de distintos tamaños con retardos.

Al igual que se ha comentado en el apartado anterior, un retardo de 150ms es indiferente frente a 70 segundos de envío.

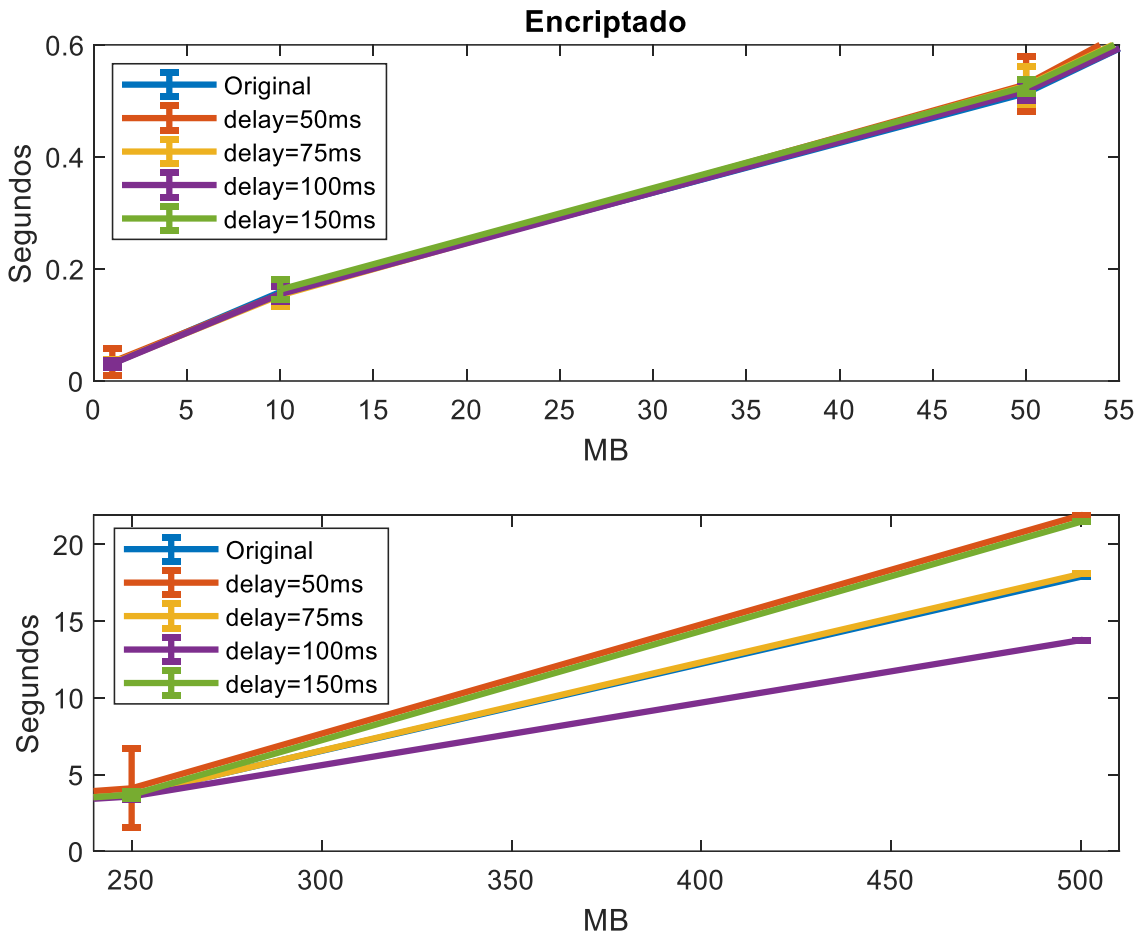

Fig. 35 Comparativa de tiempos de encriptado de un archivo de distintos tamaños con retardos.

En este caso el retardo no afecta al encriptado de archivos, ya que el retardo tiene relación con el tiempo de envío y de envío total.

5.6. ENVIAR UN ARCHIVO A UN ÚNICO SERVIDOR CON JITTER

Contexto: un cliente con un servidor (1:1). Se envía un archivo de varios tamaños

Para hacer esta prueba se ha usado la herramienta *Mininet* para simular dos terminales, cada una con su correspondiente IP. Una de ellas actuará como cliente y la otra actuará como servidor. Para añadir el *jitter* se ha utilizado el comando *jitter* de *Mininet*. Se han realizado pruebas 50ms, 100ms, 150ms y 200ms de *jitter*. [34]

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos de envío y de encriptado.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

IPOrigen 10.0.0.1

IPDestino 10.0.0.2

Datos de entrada: para esta opción los argumentos que hemos pasado han sido:

1- Opción del menú principal: enviar un archivo (1).
2- Servidor al que se desea enviar: enviar al servidor (1).
3- Archivo a intercambiar: en cada iteración se va variando el archivo.

Resultado esperado:

El objetivo es obtener un archivo con los tiempos de inicio de acción, finalizado de encriptado y finalizado de acción.

El archivo se ha recibido correctamente en destino, con el mismo tamaño y el contenido encriptado.

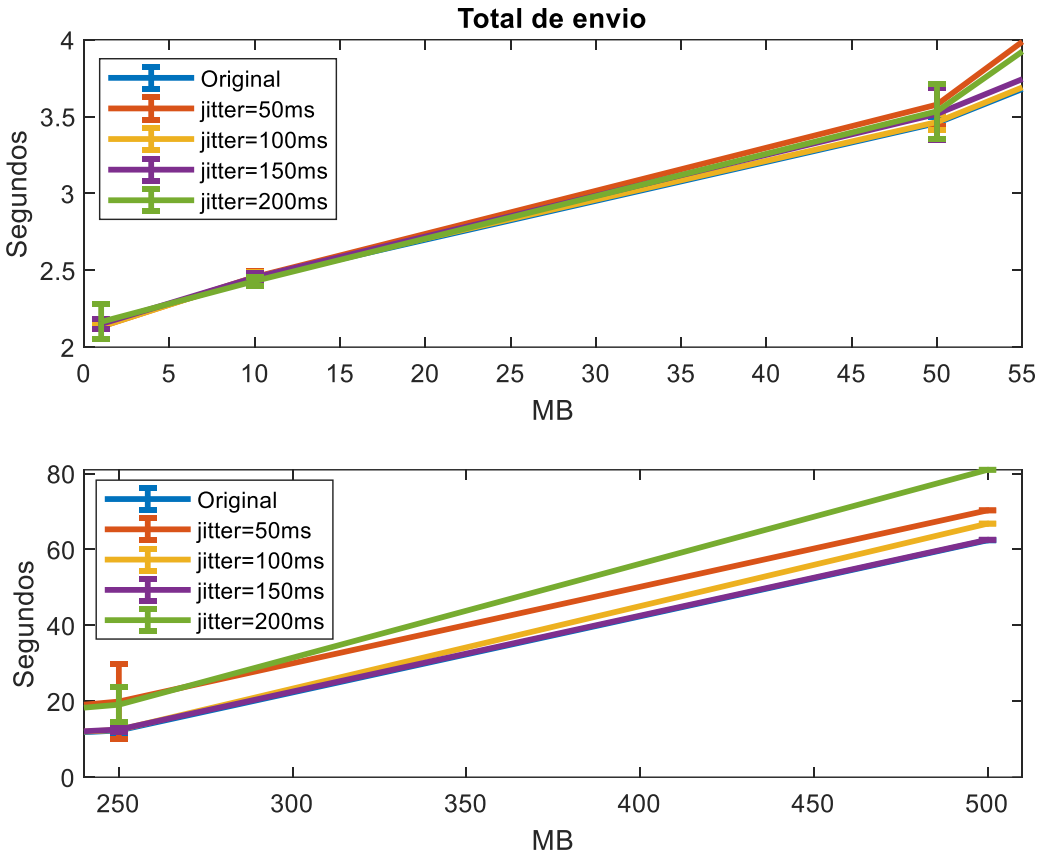

Fig. 36 Comparativa de tiempos de envío y encriptado de un archivo de distintos tamaños con jitter.

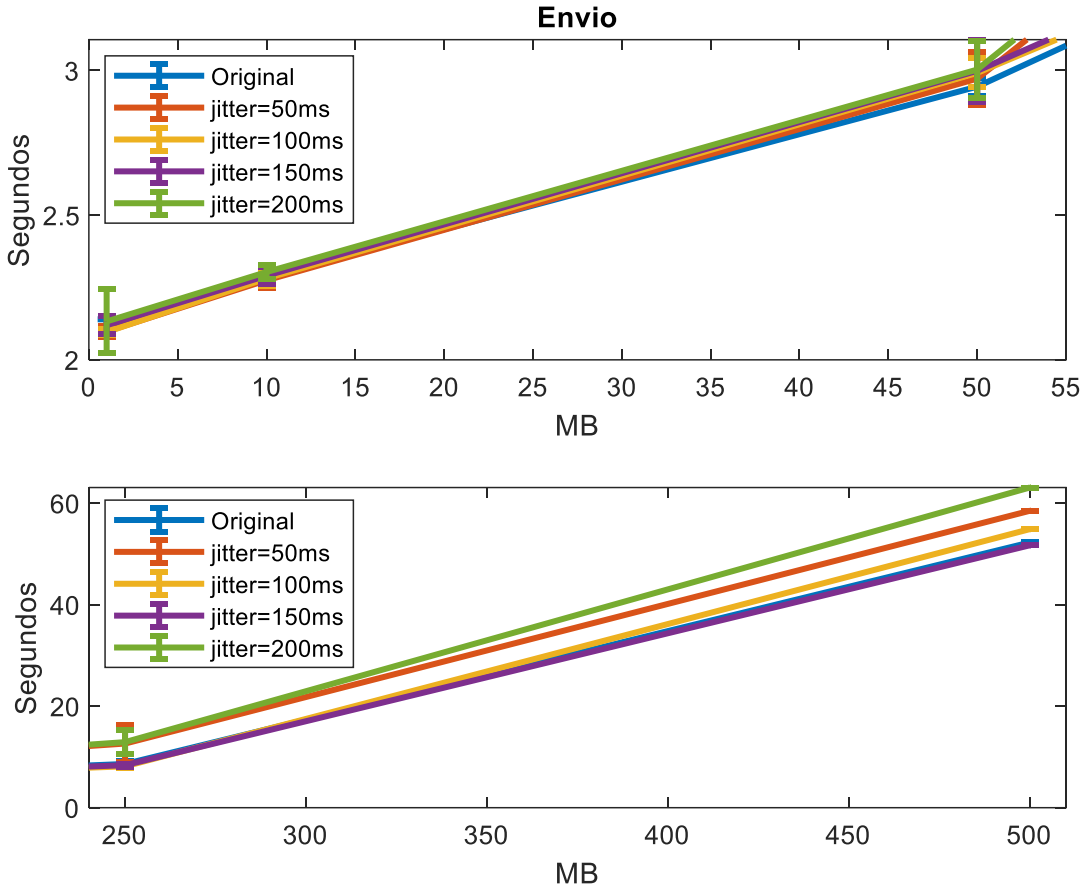

Fig. 37 Comparativa de tiempos de envío de un archivo de distintos tamaños con jitter.

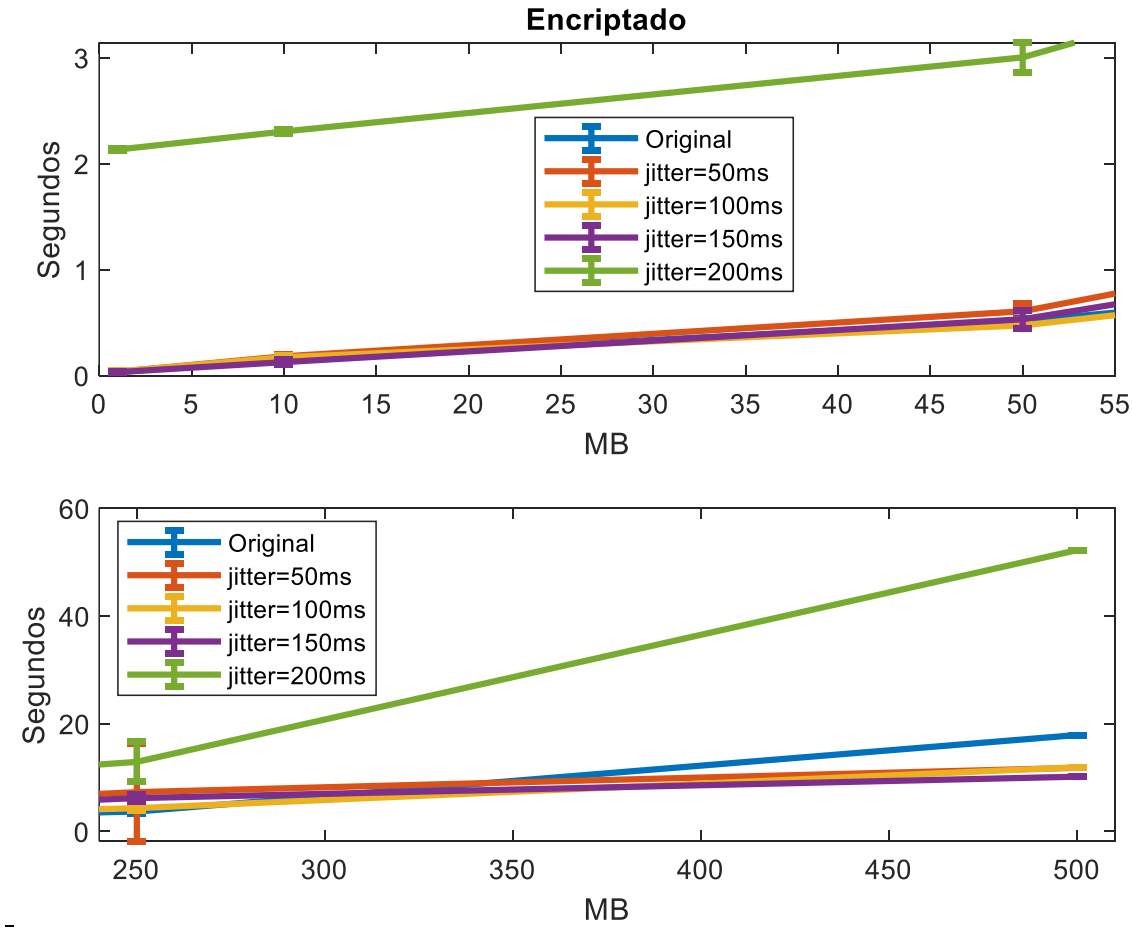

Fig. 38 Comparativa de tiempos de encriptado de un archivo de distintos tamaños con jitter.

En este caso se observa una pequeña diferencia en archivos de 500MB entre el envío original y un *jitter* de 200ms de unos 15 segundos del tiempo total.

5.7. ENVIAR UN ARCHIVO A UN ÚNICO SERVIDOR CON DISTINTO ANCHO DE BANDA

Contexto: un cliente con un servidor (1:1). Se envía un archivo de varios tamaños

Para hacer esta prueba se ha usado la herramienta *Mininet* para simular dos terminales, cada una con su correspondiente IP. Una de ellas actuará como cliente y la otra actuará como servidor. Para añadir el ancho de banda se ha utilizado el comando *bw* de *Mininet*. Se han realizado pruebas con un ancho de banda de 1MB, 10MB, 50MB, 100MB, 300MB, 1000MB. [34]

La prueba consiste en enviar un archivo de distintos tamaños con el fin de tomar tiempos de envío y de encriptado.

La prueba se ha realizado con archivos de 1MB, 10MB, 50MB, 250MB y 500MB. Para el envío del archivo de 250MB y 500MB ha sido necesario ampliar la memoria RAM utilizada por el programa, ya que si no se quedaba sin recursos.

Debido a la cantidad de MB de algunos de los archivos se ha tenido en cuenta el tiempo de cifrado de contenido en el origen y el tiempo de envío y la suma de ambos.

IPOrigen 10.0.0.1

IPDestino 10.0.0.2

Datos de entrada: para esta opción los argumentos que hemos pasado han sido:

1- Opción del menú principal: enviar un archivo (1).
2- Servidor al que se desea enviar: enviar al servidor (1).
3- Archivo a intercambiar: en cada iteración se va variando el archivo.

Resultado esperado:

El objetivo es obtener un archivo con los tiempos de inicio de acción, finalizado de encriptado y finalizado de acción.

El archivo se ha recibido correctamente en destino, con el mismo tamaño y el contenido encriptado.

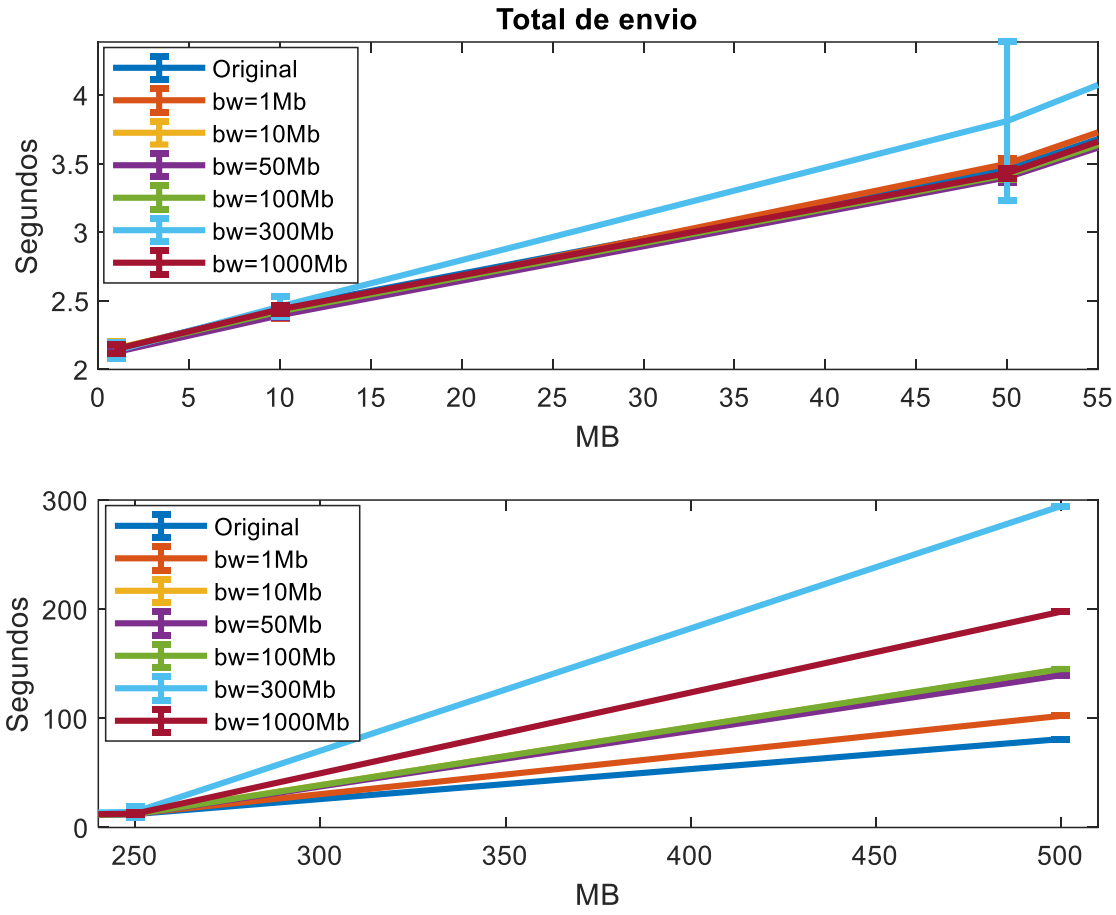

Fig. 39 Comparativa de tiempos de envío y encriptado de un archivo de distintos tamaños con distinto ancho de banda.

Estos valores variarán según el tiempo de procesado que tengan los elementos de Mininet. El original es sin ningún tipo de restricciones, por lo que irá a la tasa máxima que pueda soportar la virtualización de la red. Según se van añadiendo limitaciones se va limitando el *throughput* de TCP de manera que tardara más en trasmitir la misma información.

En el caso de 300Mb/s y 1000MB/s depende del número de retrasmisiones que sean necesarios. En caso de 300Mb/s habrá más retrasmisiones que en caso de 1000MB/s.

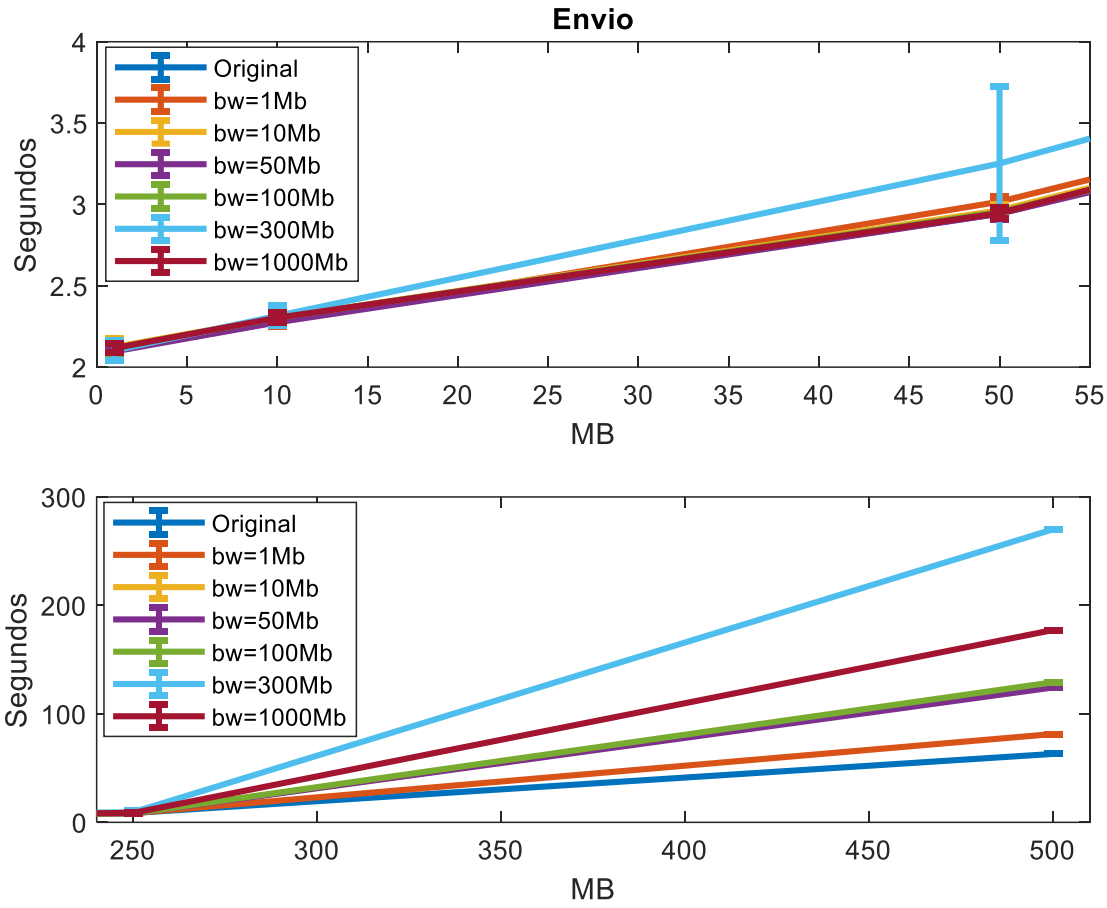

Fig. 40 Comparativa de tiempos de envío de un archivo de distintos tamaños con distinto ancho de banda.

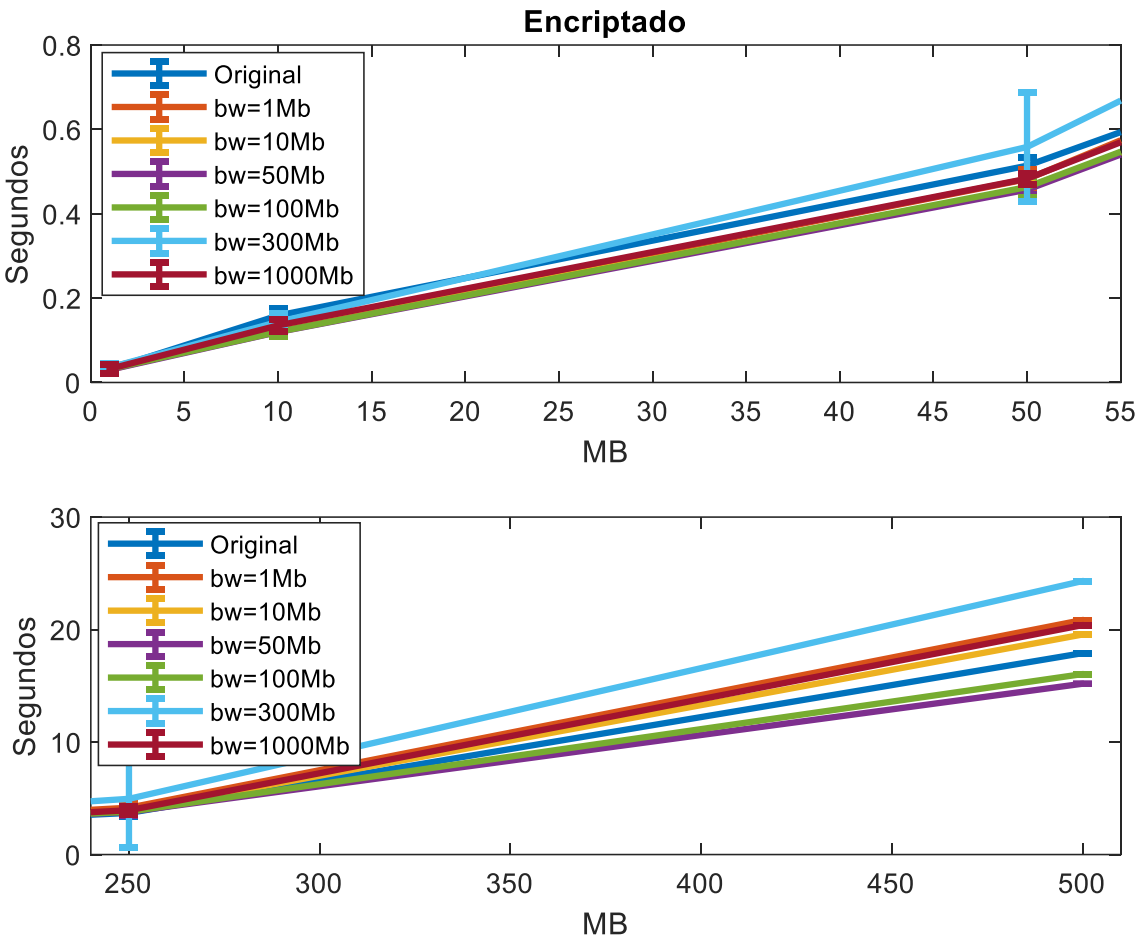

Fig. 41 Comparativa de tiempos de encriptado de un archivo de distintos tamaños con distinto ancho de banda.

5.8. COMPARATIVA DE TIEMPOS CON SERVIDORES DE AMAZON, AZURE Y MICROSOFT

Para realizar esta prueba se ha hecho ping desde el cliente a un servidor y desde un cliente a todos los servidores. Los tiempos de envío al tratarse de un entorno no local, la media no supera los 0.2 ms.

En figura 45 se observa que los tiempos de RTT entre los distintos servidores de *Amazon Web Service* y *Azure* en el mejor de los casos es de 0 ms, sin embargo, al trabajar en un entorno más local, la media de los *RTTs* entre los nodos de Mininet es de media 0.158 ms, como vemos en la figura 42.

En el peor de los casos con *RTT* entre los servidores más lejanos vemos un RTT de 500 ms, muy por encima de nuestro peor caso, que no llega a 1 ms.

En la prueba que se ha realizado se ha hecho ping a California y a Holanda para determinar los RTT en distintos escenarios. Para ello se han realizado pruebas desde un ordenador portátil, conectado a una red Wi-Fi y por cable Ethernet. Se ha repetido la prueba desde un dispositivo móvil, conectado a

una red Wi-Fi y con el uso de datos móviles. Los tiempos obtenidos en la Figura 43 se ven mejorados en caso de que se enviaran los archivos a un entorno local.

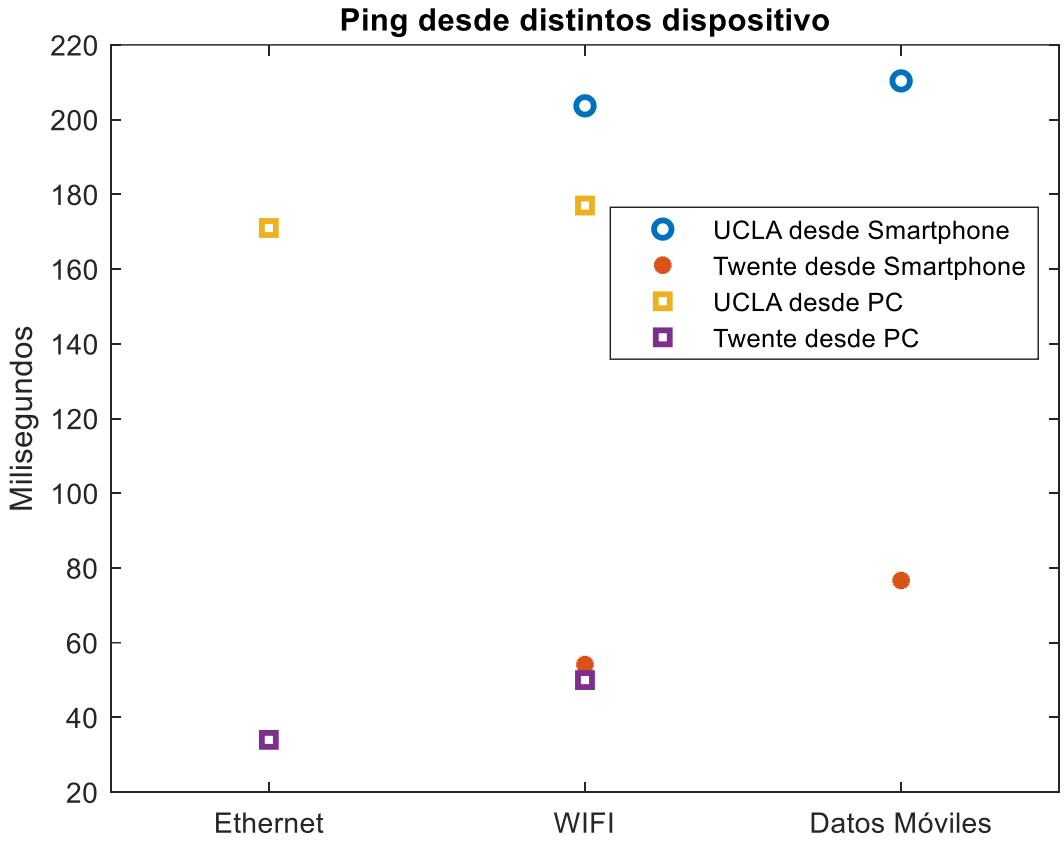

Fig. 42 Ping desde distintos dispositivos.

Ping www.google.com

Dispositivo móvil con Wifi	15-20 ms
Dispositivo móvil con datos	50ms
Ordenador con Wi-Fi	4 ms

Tabla V Tiempos ping a www.google.com.

Se observa que los tiempos desde un dispositivo móvil aumentan unos 15 segundos cuando está conectado a una red Wi-Fi y entre 50-150 ms con un dispositivo móvil conectado a una red de datos

De esta forma, comprobamos que al trabajar en un entorno *Fog Computing*, los *RTTs* mejoran notablemente comparado con un *Cloud Computing*.

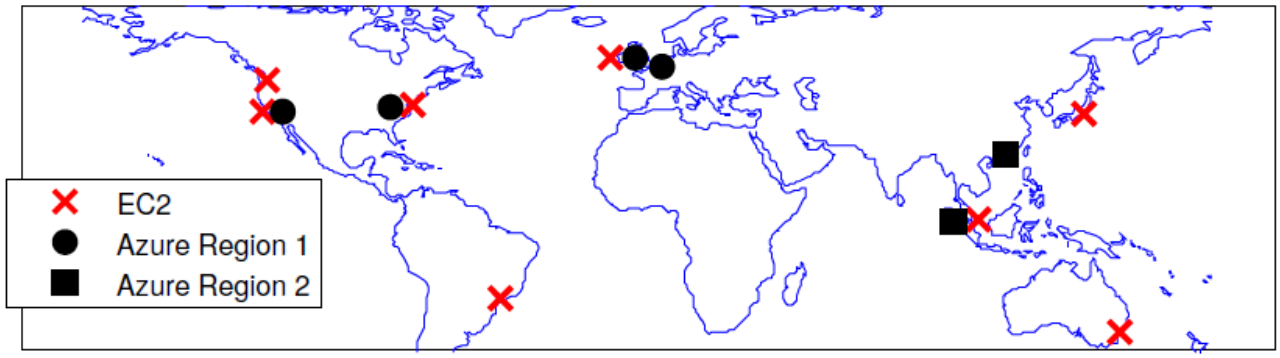

Fig. 43 Ubicación servidores Cloud [35].

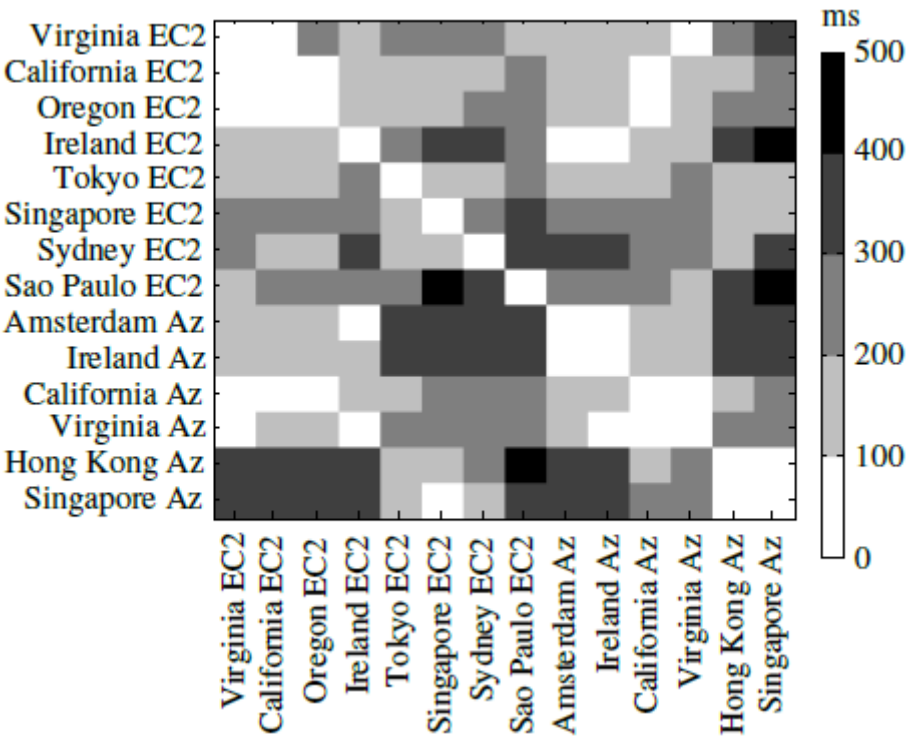

Fig. 44 RTT entre distintos data-centers (milisegundos) [35].

6. CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se va a hacer una valoración general del trabajo llevado a cabo. Primero se analizará si se han cumplido los resultados esperados antes de empezar con el trabajo. Se indicará cual ha sido el aprendizaje durante la realización del mismo. Y, por último, se propondrán futuras mejoras del proyecto.

6.1. CONCLUSIONES

El objetivo del proyecto era el diseño e implementación de un entorno *Fog Computing*, con el fin de crear un entorno de almacenamiento más cercano al usuario, sin tener que subir los datos a la nube. Al tratar la información desde un entorno más local, disminuye la latencia y el consumo de ancho de banda.

Se ha desarrollado un entorno cliente servidor en java que permita el intercambio de archivos de forma segura.

Para una mayor seguridad, la información se divide entre varios servidores, siendo el cliente el único que puede recuperar los fragmentos del archivo.

Se ha desarrollado un *blockchain* para tener el historial de archivos compartidos en cada uno de los nodos.

Las pruebas se han realizado con distintos archivos. Observamos que se envía un archivo de 500MB en menos de 10 segundos, por lo que el entorno *Fog* no ralentiza el envío.

Se puede consultar el programa en el siguiente enlace, en el que se encuentra el archivo README.md con los pasos a seguir para lanza el proyecto.

`https://github.com/dbahon/fogComputing`

6.2. TRABAJO FUTURO

Como posibles mejoras del trabajo se proponen las siguientes:

- Trabajar con mayor cantidad de nodos para observar resultados más claros.
- Envío de información duplicada a distintos servidores. Con el fin de evitar posibles pérdidas de información.
- Llevar el desarrollo a dispositivos móviles, con el fin de observar los resultados con mayor claridad.
- Realización de pruebas con distintos tamaños de archivos, que ha sido difícil realizar debido a las limitaciones del ordenador donde se estaban llevando a cabo las pruebas.
- Envío de archivos de forma paralela para mejorar los tiempos.

REFERENCIAS

[1] Saúl Alonso-Monsave, Félix García-Carballeira y Alejando Calderón, «Fog Computing Through Public-Resource Computing and Storage,» *Second International Conference on Fog and Mobile Edge Computing (FMEC),* pp. 81-87, 2017.

[2] Shanhe Yi, Cheng Li y Qun Li, «A Survey of Fog Computing: Concepts, Applications and Issues,» *Proceedings of the 2015 Workshop on Mobile Big Data,* pp. 37-42, 2015.

[3] Osanaiye Opeyemi, Shuo Chen, Zheng Yan, «From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework,» pp. 8284 - 8294, 2017.

[4] Kirak Hong, David Lillethum, Umakishore Ramachandran, Beate Ottenwälder y Boris Koldehofe, «Mobile Fog: A Programming Model for Large-Scale Applications on the Internet of Things,» *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing,* pp. 15-20, 2013.

[5] Stephane Caron, Frederic Giroire, Dorian Mazauric, Julian Monteiro, Stéphane Pérennes, «P2P storage systems: Study of different placement policies,» *Peer-to-Peer Networking and Applications,,* pp. 427-443, 2013.

[6] Luis M.Vaquero, Luis Rodero-Merino, «Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing,» *Computer Communication Review,* pp. 27-32, 2014.

[7] «https://www.makesoft.es/es/breve-historia-del-cloud-computing/,» 02 08 2018. [En línea]. [Último acceso: 20 Septiembre 2018].

[8] Lizhe WANG, Gregor von LASZEWSKI, Andrew YOUNGE, Xi HE, «Cloud Computing: a Perspective Study,» Rochester, New York, 2010.

[9] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov, «The Eucalyptus Open-source Cloud-computing System,» pp. 124-131.

[10] Goran Čandrlić, «globaldots,» 19 Marzo 2013. [En línea]. Available: https://www.globaldots.com/cloud-computing-types-of-cloud/. [Último acceso: 18 Septiembre 2018].

[11] «Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-iaas/. [Último acceso: 18 Septiembre 2018].

[12] Yang, Haibo and Tate, Ma, «A Descriptive Literature Review and Classification of Cloud Computing Research,» de *Communications of the Association for Information Systems*, vol. 31, 2012.

[13] «Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-paas/. [Último acceso: 18 Septiembre 2018].

[14] «Microsoft Azure,» [En línea]. Available: https://azure.microsoft.com/es-es/overview/what-is-saas/. [Último acceso: 18 Septiembre 2018].

[15] Peter Mell, Timothy Grance, «The NIST Definition of Cloud Computing,» *National Institute of Standards and Technology*, p. 7, 2011.

[16] Yashpalsinh Jadeja,Kirit Modi, «Cloud Computing - Concepts, Architecture and Challenges,» de *International Conference on Computing, Electronics and Electrical Technologies* , 2012.

[17] Feng Xia, Laurence T. Yang, Lizhe Wang and Alexey Vinel, «Internet of Things,» *INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS*, pp. 1101-1102, 2012.

[18] «theguardian,» 10 2003. [En línea]. Available: https://www.theguardian.com/technology/2003/oct/09/shopping.newmedia. [Último acceso: 20 Septiembre 2018].

[19] Felix Wortmann, Kristina Flüchter, «Internet of Things. Technology and Value Added,» pp. 221-224.

[20] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, Sateesh Addepalli, «Fog Computing and Its Role in the Internet of Things,» *Cisco Systems Inc*, pp. 13-15, 2012.

[21] I. Errando, Interviewee, *Fog computing.* [Entrevista].

[22] Songze Li, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr, «Coding for Distributed Fog Computing,» *IEEE Communications Magazine*, pp. 34-40, Abril 2017.

[23] Shouddy Tárano León, Tatiana Delgado Fernández, Alejandro Luar Pérez Colomé, «Towards smarter cities taking advantage of the Fog Computing paradigm,» Universidad Tecnológica de La Habana, CUJAE - Cuba, 2018.

[24] Eusebio Antonio Martínez Santos, Mayo 2008. [En línea]. Available: http://ealpj.blogspot.com/2008/05/ealpj.html. [Último acceso: 20 Septiembre 2018].

[25] B. Valencia, S. Santacruz, L. Becerra y J.J. Padilla,Y., «Mininet: una herramienta versátil para emulación y prototipado de Redes Definidas por Software,» 2015.

[26] Jackson Emilio Martínez Copete, «Estudio del funcionamiento de la herramienta Mininet,» 2015.

[27] «boxcryptor,» [En línea]. Available: https://www.boxcryptor.com/es/encryption/. [Último acceso: 02 08 2018].

[28] Panu Hämäläinen, Timo Alho, Marko Hännikäinen, and Timo D. Hämäläinen, «Design and Implementation of Low-area and Low-power,» Tampere University of Technology, 2006.

[29] Gregorio Díaz, Fernando Cuartero, Valentín Valero and Fernando Pelayo, «Automatic Verification of the TLS HandShake Protocol,» *Symposium on Applied Computing*, pp. 789-794, 2004.

[30] William Mougayar, «oreilly,» 16 01 2015. [En línea]. Available: https://www.oreilly.com/ideas/understanding-the-blockchain. [Último acceso: 20 Septiembre 2018].

[31] Carlo D'agostino, «Club de innovacion,» [En línea]. Available: http://www.clubdeinnovacion.com/bloginn/blockchain-publico-privado. [Último acceso: 27 Octubre 2018].

[32] Juan Benet, «IPFS - Content Addressed, Versioned, P2P File System,» 2014.

[33] «java-buddy,» 20 07 2016. [En línea]. Available: http://java-buddy.blogspot.com/2016/07/java-example-of-ssl-server-and-client.html. [Último acceso: 2018 08 08].

[34] Mininet, «Mininet,» 2018. [En línea]. Available: http://mininet.org/walkthrough/. [Último acceso: 20 Septiembre 2018].

[35] José Luis García Dorado, Sanjay G. Rao, «Cost-aware Multi Data-Center Bulk Transfers in the Cloud From a Customer-Side Perspective,» 2015.

[36] David Morales, «Academia,» [En línea]. Available: http://www.academia.edu/8826530/TUTORIAL_MININET. [Último acceso: 2018 Noviembre 2].